Universidade do Minho
Escola de Engenharia

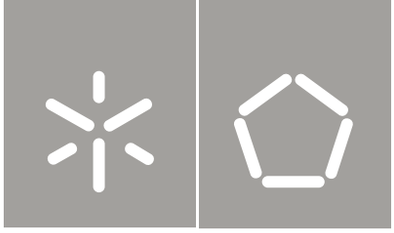
Rui Miguel Costa Rodrigues

Router Dual-Radio para Redes de Sensores sem Fios

Rui Miguel Costa Rodrigues *Router Dual-Radio* para Redes de Sensores sem Fios

UMinho | 2013

outubro de 2013



Universidade do Minho
Escola de Engenharia

Rui Miguel Costa Rodrigues

Router Dual-Radio para Redes de Sensores sem Fios

Tese de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia de Comunicações

Trabalho efetuado sob a orientação do
Professor Doutor José Augusto Afonso

outubro de 2013

Agradecimentos

Gostaria de agradecer ao professor José Afonso, pela orientação, dedicação e paciência durante a realização deste trabalho.

Aos meus pais por tudo.

À Sandra pelo apoio e pela paciência nos bons e nos maus momentos.

Aos meus colegas de laboratório e de curso pelo apoio dado no desenvolvimento deste trabalho.

Resumo

As redes de sensores sem fios são cada vez mais utilizadas, tanto na indústria como em aplicações domésticas, muito devido ao grande desenvolvimento que têm sofrido nos últimos anos. Este desenvolvimento tem levado a que surjam novos sensores de baixo consumo, baixo débito, e baixo custo. A informação adquirida pelos sensores precisa de ser enviada para outros nós da rede (e.g., uma estação base ou um atuador). Em redes ZigBee, os *routers* tem a função de reencaminhar a informação para o destino, em topologias *multihop*, como malha e árvore. Isto tem um impacto sobre o desempenho, principalmente em aplicações que geram um tráfego elevado como é o caso do transporte de informação de eletrocardiografia (ECG), eletroencefalografia (EEG) ou postura (captura de movimentos).

Nesta dissertação é feito o estudo, desenvolvimento e teste de um *router dual-radio* 802.15.4/802.15.4 com o objetivo do aumento de desempenho de redes IEEE 802.15.4 e ZigBee, ou a extensão de protocolos *single-hop*, como o LPRT, para topologias *multihop*. A implementação é feita utilizando plataformas de hardware e software da Texas Instruments. Inicialmente é feita a análise do desempenho de uma rede ZigBee com topologia em estrela e com topologia em árvore com dois saltos, e da comunicação SPI entre dois dispositivos com vista a verificar o modo de comunicação mais eficiente. Com base nesta análise é implementado um *router dual-radio* que permite um aumento de 51% do débito máximo em relação ao uso de um *router* normal, conforme validam os resultados experimentais obtidos.

.

Abstract

Wireless sensor networks are growing widely, in industry and domestic applications, largely due the advances achieved in the last years. This development had led to the emergence of new sensors with low power, low rate, and low cost. The information acquired by the sensors need to be sent to other network nodes (e.g., a base station or an actuator). On ZigBee networks, routers are used to forward the information to the destination in multihop topologies, such as mesh and tree. Normal routers typically have a single transceiver. This has an impact on performance, especially in applications that generate high traffic such as transport of electrocardiogram (ECG), electroencephalography (EEG) or posture (motion capture) information.

This dissertation presents the study, development and testing of a dual-radio router 802.15.4/802.15.4 with the goal of increasing performance of IEEE 802.15.4 and ZigBee networks, or the extension of single-hop protocols, such as LPRT, to multihop topologies. The implementation is achieved by using hardware and software platforms from Texas Instruments. Firstly, it is made the performance analysis of ZigBee networks with star and 2-hop tree topology, and then the evaluation of the SPI communication between two devices in order to verify the most efficient communication mode. Based on this analysis it is implemented a dual-radio router that allows an increase of 51% in the maximum throughput compared to a normal router, as the experimental results show.

Índice de conteúdos

Lista de figuras

Lista de tabelas

Lista de abreviaturas

ACK	Acknowledgement
ADC	Analog-to-digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
APS	Application Support Sublayer
BER	Bit Error Rate
BPSK	Binary Phase-Shift Keying
CCA	Clear Channel Assessment
CDMA	Code Division Multiple Access
CPHA	SPI Clock Phase
CPOL	SPI Clock Polarity
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
ECG	Eletrocardiograma
EEG	Eletroencefalograma
ETSI	European Telecommunications Standards Institute
FAB	Frame Alignment Byte
FFD	Full Function Device
GSM	Global System for Mobile Communications
GTS	Guaranteed Time Slot
IEEE	Institute of Electrical and Electronics Engineers

IETF	Internet Engineering Task Force
IP	Internet Protocol
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization
ITU	International Telecommunications Union
LAN	Local Area Network
LCD	Liquid-crystal Display
LEACH	Low-Energy Adaptive Clustering Hierarchy
LED	Light-emitting Diode
LPRT	Low-Power Real Time
LSB	Least Significant Bit
M2M	Machine to Machine
MAC	Medium Access Control
MSN	Most Significant BIT
NLDE	NWK Layer Data Entity
NLME	NWK Layer Management Entity
NPDU	NWK Packet Data Unity
NPDU-SAP	NPDU Service Access Point
NWK	Network
O-QPSK	Offset Quadrature Phase shift Keying
PAN	Personal Area Network
PDU	Protocol Data Unit
RAM	Read-only Memory
RDR	Router Dual-Radio

SAP	Service Access Point
SoC	System-on-Chip
SPI	Serial Peripheriral Bus
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WPAN	Wireless Personal Area Network
WWAN	Wireless Wide Area Network
WSN	Wireless Sensor Network
ZC	ZigBee Coordinator
ZED	ZigBee End Device

1. Introdução

1.1 Enquadramento e motivação

O desenvolvimento tecnológico dos últimos anos tem introduzido avanços significativos nos sistemas de telecomunicações. No tempo atual as comunicações sem fios tornaram-se cada vez mais na principal opção para implementação de sistemas de comunicações. De entre os vários tipos de redes sem fios, as redes de sensores sem fios são as que mais evoluíram. A necessidade de uma constante monitorização, aliada à eficiência energética, são algumas das características que estas redes apresentam. As redes de sensores sem fios servem como um interface para o mundo real, permitindo obter informações físicas como temperatura, quantidade de luz, radiação, etc. e o seu processamento por um sistema computacional. Nas redes de sensores sem fios todos os dispositivos colaboram para um objetivo comum, podendo as suas ligações serem alteradas de acordo com um contexto específico, sem a necessidade de uma infraestrutura de suporte.

O recente avanço nas áreas de tecnologias de informação, comunicações sem fios, e microeletrónica está a abrir caminho a uma nova geração de dispositivos sensores e atuadores de baixo custo permitindo criar soluções de monitorização com grande escala de resolução e precisão, que antes não estavam disponíveis. Este desenvolvimento dos sensores fez surguir um novo leque de aplicações para a saúde, meio ambiente, energia, segurança alimentar, proteção e segurança de infraestruturas, aplicações industriais, agricultura, *smartphones*, comunicações M2M, etc. Este desenvolvimento tem levado a que surjam novos sensores de baixo consumo, baixo débito, e baixo custo.

Numa rede de sensores sem fios, a informação adquirida pelos sensores precisa de ser enviada para outros nós da rede (e.g., uma estação base ou um atuador). Em redes ZigBee [3], os *routers* tem a função de reencaminhar a informação para o próximo ponto do destino. Os *routers* normais possuem tipicamente um único

transceiver. Em topologias *multihop*, como malha e árvore, isto tem um impacto sobre o desempenho, principalmente em aplicações que geram um tráfego elevado como é o caso do transporte de informação de eletrocardiografia (ECG), eletroencefalografia (EEG) ou postura (captura de movimentos), porque durante o processo de transmissão de um pacote o *router* tende a rejeitar outros pacotes que lhe estejam a ser enviados.

Esta dissertação visa a conceção, desenvolvimento e teste de um *router* dual-*radio* 802.15.4/802.15.4 com o objetivo do aumento de desempenho de redes IEEE 802.15.4 e ZigBee, ou a extensão de protocolos *single-hop*, como o LPRT [16], para topologias *multihop*. Este *router dual-radio* (RDR) será formado por dois dispositivos ligados entre si por uma ligação física (SPI), que serão programados para operar em canais diferentes. A utilização do RDR permitirá o aumento do débito pois, como são utilizados canais diferentes para enviar e receber, haverá menos colisões no acesso ao meio.

1.2 Objetivos

Após um levantamento teórico das necessidades do projeto foram enumerados os objetivos principais:

- Análise teórica e prática do desempenho de uma rede ZigBee em estrela e em árvore com dois saltos;
- Implementação das várias formas de comunicação SPI entre módulos CC2530 com vista a obter a comunicação mais eficiente;
- Desenvolvimento e integração do RDR, teste e análise da sua performance.

1.3 Estrutura da dissertação

No capítulo 1 é feita uma breve introdução ao tema desta dissertação. É descrita a motivação para a realização deste trabalho juntamente com os respetivos

objetivos a alcançar. Este capítulo termina com a descrição da estrutura adotada para esta dissertação.

O capítulo 2 apresenta o estado da arte relativamente a protocolos de comunicação e redes de sensores sem fios. É feita uma introdução à norma 802.15.4 e à especificação ZigBee. Este capítulo apresenta também algumas considerações teóricas referentes ao trabalho efetuado.

No capítulo 3 é feita uma descrição geral do sistema, seguida de uma descrição detalhada sobre o hardware utilizado. É feita depois a análise teórica e prática à obtenção do débito máximo entre dois dispositivos (topologia em estrela) e utilizando um *router* intermédio (topologia em árvore com dois saltos).

No capítulo 4 é abordada a comunicação SPI entre dois módulos CC2530 [19]. Para determinar a implementação mais eficiente são apresentados os testes aos diferentes modos de processamento do SPI pelo CC2530. O modo mais eficiente é depois utilizado no capítulo 5 para o desenvolvimento do RDR.

Seguidamente, no capítulo 5, é feita a descrição e implementação do RDR e apresentados os valores dos débitos obtidos. É depois feita uma comparação de desempenho entre o RDR e um *router* normal.

Por fim, o capítulo 6 apresenta as conclusões desta dissertação, resumindo os objetivos cumpridos, as possíveis modificações a efetuar ao sistema, e propostas de trabalho futuro.

2. Estado da arte

2.1 Introdução

Uma rede sem fio (w*ireless*) é um sistema que interliga vários equipamentos fixos ou móveis utilizando ondas mecânicas (e. g., ultrassom) ou eletromagnéticas (e. g., luz, ondas rádio) como meio de transmissão. As redes sem fios são maioritariamente sistemas baseado em ondas rádio, sendo que cada rede é configurada para funcionar numa dada frequência, ou grupo de frequências, do espectro eletromagnético. O espectro e as normas das comunicações sem fios são geridos pela União internacional das telecomunicações (ITU), sendo cada país, com as suas agências, responsável pela sua regulamentação. O espectro rádio tem alocado diversas frequências licenciadas para diferentes tecnologias de comunicações, ex. GSM ou radionavegação. Um grupo de frequências livres de licença foi atribuída para aplicações industriais, científicas e médicas (ISM) [17], e.g. Wi-Fi [20] ou Bluetooth [21]. Esta gama é apresentada na Tabela 2.1, juntamente com a sua disponibilidade mundial.

Tabela 2.1 - Gama de frequência ISM e a sua disponibilidade.

Gama de Frequências		Frequência central	Disponibilidade
6.765 MHz	6.795 MHz	6.780 MHz	Sujeito a aceitação local
13.553 MHz	13.567 MHz	13.560 MHz	Mundial
26.957 MHz	27.283 MHz	27.120 MHz	Mundial
40.660 MHz	40.700 MHz	40.680 MHz	Mundial
433.050 MHz	434.790 MHz	433.920 MHz	Sujeito a aceitação local
902.000 MHz	928.000 MHz	915.000 MHz	Sujeito a aceitação local
2.400 GHz	2.500 GHz	2.450 GHz	Mundial
5.725 GHz	5.875 GHz	5.800 GHz	Mundial
24.000 GHz	24.250 GHz	24.125 GHz	Mundial
61.000 GHz	61.500 GHz	61.250 GHz	Sujeito a aceitação local
122.000 GHz	123.000 GHz	122.500 GHz	Sujeito a aceitação local
244.000 GHz	246.000 GHz	245.000 GHz	Sujeito a aceitação local

Estas frequências têm a vantagem de não ser necessária licença para o seu uso, ao contrário das bandas alocadas para um determinado serviço de comunicações pago.

Por outro lado, diversas tecnologias sem fios usam as mesmas gamas de frequências ISM, como por exemplo o IEEE 802.15.4, Bluetooth, Wi-Fi e aplicações privadas ou tecnologias em desenvolvimento. Isto pode trazer um impacto negativo porque, caso algumas tecnologias partilhem o mesmo meio físico e a mesma frequência, o nível de interferência pode aumentar significativamente, o que causará uma degradação da performance destas redes.

As redes sem fios podem ser classificadas de acordo com a sua área de abrangência: redes pessoais ou curta distância (WPAN), redes locais (WLAN), redes metropolitanas (WMAN) e redes geograficamente distribuídas ou de longa distância (WWAN). O diagrama apresentado na Figura 2.1 apresenta os diferentes tipos de redes sem fios em função do débito e da mobilidade.

As redes sem fios de área pessoal (WPANs) são utilizadas para troca de informação entre dispositivos a distâncias relativamente pequenas. Ao contrário das redes de área local (WLANs), as WPANs não envolvem infraestruturas de rede muito complexas, o que permite que possam ser desenvolvidas soluções económicas para um vasto leque de dispositivos. Isto permite implementar soluções pequenas, de custo reduzido e com eficiência energética para uma grande diversidade de dispositivos.

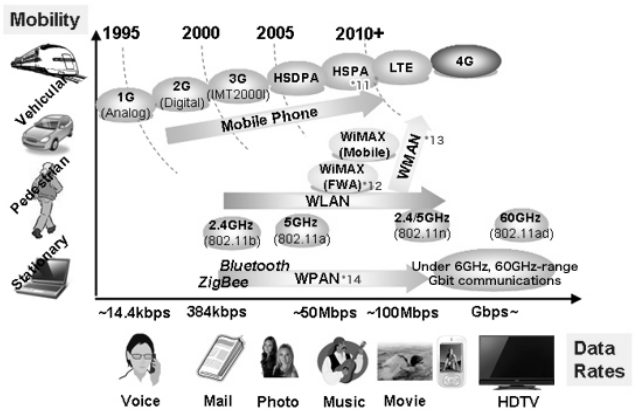


Figura 2.1 – Diagrama com redes sem fios em função do débito e mobilidade [18].

2.2 Redes de sensores sem fios

As redes de sensores sem fios (WSN) são um subgrupo das redes sem fios focado em permitir conectividade sem fios a sensores a atuadores em geral. Estas podem ser classificadas de acordo com o tipo de sensor, tipo de aplicação (industrial, médica, veicular, etc.), parâmetros monitorizados (vibração, aceleração, temperatura, etc.), e pelos parâmetros da rede (topologia, débito, alcance, etc.). As principais características de uma rede de sensores sem fios são [2]:

- Baixa complexidade de dispositivos;
- Baixa taxa de dados;
- Baixo consumo de energia;
- Capacidade de lidar com falhas de nós da rede;
- Comunicações *multihop*;
- Escalabilidade;
- Capacidade de funcionamento em ambientes extremos;
- Operação ad-hoc.

Uma WSN possui uma série de requisitos, como a capacidade de fornecer informação sobre o meio onde está inserida, detetar eventos, robustez (possuir tolerância a falhas), que tenha um ciclo de vida o mais longo possível, que seja escalável de modo a suportar um número de nós elevado, e que seja flexível para modificar as tarefas, quer diretamente, quer através da própria WSN.

Os nós da rede podem ser vistos como pequenos computadores, com interfaces e componentes bastante básicos. Geralmente consistem numa unidade de processamento com capacidade computacional limitada e memória limitada, sensores, um dispositivo de comunicações (normalmente um *radio transceiver*), e uma fonte de alimentação, como apresentado na Figura 2.2.

Os fabricantes dos nós inicialmente incluíam os componentes separados, como o caso do microcontrolador e do rádio. Mais recentemente têm optado pela solução

system-on-chip (SoC) onde estes componentes estão integrados num único chip, permitindo uma diminuição do tamanho dos dispositivos da WSN.

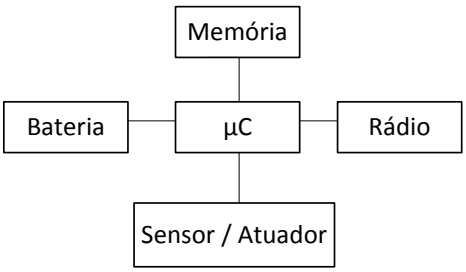


Figura 2.2 – Estrutura básica de um nó de uma rede de sensores sem fios.

2.3 IEEE 802.15.4

O IEEE 802.15.4 [2] é uma norma para as camadas física e de acesso ao meio, em redes de área pessoal sem fios de baixo débito. O alvo principal desta norma são as aplicações que não necessitem de taxas de transferência elevadas, tais como aplicações de controlo e monitorização. Esta especificação tem como objetivos maximizar a longevidade dos dispositivos alimentados por baterias e minimizar a complexidade de modo a reduzir os custos de hardware. É a base para as especificações ZigBee [6], WirelessHart [7], e MiWi [8]. Estas especificações definem uma extensão à norma, desenvolvendo as camadas superiores, que não estão definidas no IEEE 802.15.4.

As principais características de uma rede de sensores sem fios de baixo débito (LR-WPAN) são:

- Taxas de transmissão de 250 kbps, 40 kbps e 20 kbps;
- Operação em redes com topologia em estrela e *peer-to-peer*;
- Endereço de rede e endereço MAC (Media Access Control);
- Alocação de *slots* de tempo (GTS - Guaranteed Time Slots);
- Acesso ao canal através de *Carrier Sense Multiple Access With Collision Avoidance* (CSMA-CA);
- Uso de *Acknowledgements* (ACK) para garantir fiabilidade;
- Baixo consumo energético;

- Deteção de energia (ED);
- Indicação da qualidade da ligação.
- 16 Canais na banda dos 2450 MHz, 10 canais na banda de 915 MHz e 1 canal na banda 868 MHz

As redes 802.15.4 são constituídas por dois tipos de dispositivos: *full-function device* (FFD) e r*educed-function device* (RFD). Os dispositivos FFD são dispositivos que permitem realizar todas as funções enquanto dispositivos RFD permitem realizar apenas algumas. Os FFD podem ser coordenadores da rede ou apenas dispositivos, e comunicam para todos os dispositivos na rede. Os RFD são dispositivos com o propósito de serem utilizados em aplicações simples, como por exemplo, *switch* que controla uma lâmpada, e podem comunicar unicamente com dispositivos FFD.

2.3.1 Topologia da rede

Dependendo da aplicação, o IEEE 802.15.4 pode operar em dois tipos de topologias de rede: estrela ou *peer-to-peer*. Na topologia estrela existe um dispositivo central com a função de controlar a rede, o coordenador PAN. Para além de neste dispositivo poder estar implementada uma aplicação, tem a função de criar e terminar a rede. Funciona também como encaminhador de informação de um dispositivo para outro na rede. O coordenador é considerado o controlador principal da rede. Cada um dos dispositivos deve ter dois endereços para os identificar na rede, um de 64 bits, que permite a comunicação direta na PAN, e um endereço de rede de 16 bits.

Na topologia *peer-to-peer* existe, tal como na topologia em estrela, um coordenador (PAN), e qualquer dispositivo pode comunicar diretamente com outro sem que a informação tenha que passar obrigatoriamente por ele. Este tipo de topologia permite construir redes mais complexas, como por exemplo, redes com topologia em malha, e aumentar o alcance através de múltiplos saltos entre os dispositivos. Cada uma das PANs seleciona um identificador único que permite a comunicação com dispositivos na mesma rede e entre dispositivos de PANs diferentes, tendo neste caso a informação que passar pelo coordenador da PAN.

A Figura 2.3 apresenta um diagrama com as ligações para as topologias suportadas pelo IEEE 802.1.5.4.

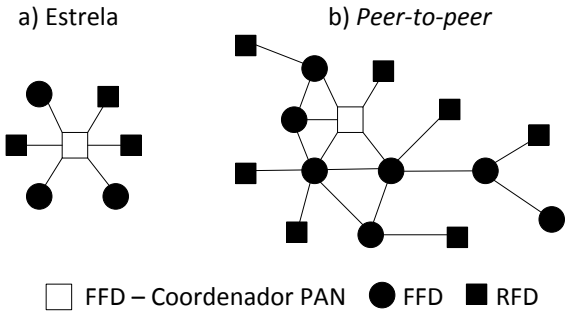


Figura 2.3 – Topologia em estrela (a) e *peer-to-peer* (b).

Uma rede em estrela pode ser criada por qualquer dispositivo FFD, onde este escolhe um identificador (PAN ID) que não esteja a ser usado noutra PAN existente no mesmo canal de comunicação. A partir deste momento pode-se juntar à rede dispositivos quer sejam FFD ou RFD.

Numa rede com topologia *peer-to-peer* qualquer dispositivo pode comunicar diretamente com outro na rede. Um exemplo de uma rede deste tipo é a *cluster-tree*, apresentada na Figura 2.4. Cada um dos FFDs pode atuar como coordenador de uma PAN, onde existe um coordenador de uma PAN a partir do qual toda a rede foi inicialmente criada. Só os dispositivos FFD permitem efetuar associações visto que os RFDs são considerados pontos terminais da rede. Cada coordenador emite *beacons* e qualquer dispositivo que receba pode pedir para se juntar. Caso o pedido seja aceite, o coordenador adiciona o novo dispositivo à sua lista de vizinhos. O dispositivo que se juntou adiciona o coordenador como seu "pai" à sua lista de vizinhos.

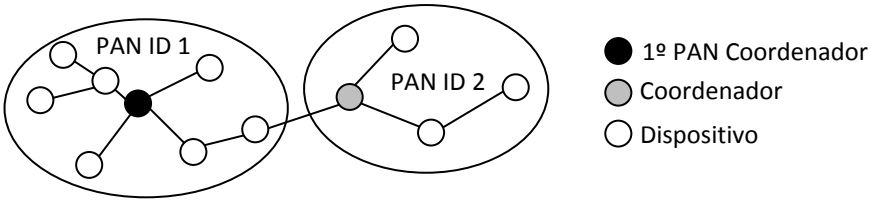


Figura 2.4 - Topologia de rede em *cluster-tree* [10].

2.3.2 Arquitetura

A arquitetura do IEEE 802.15.4 é baseada no modelo OSI, onde cada camada é responsável por uma parte das funcionalidades da norma e disponibiliza serviços para a camada superior. Desta maneira torna-se mais simples de implementar o protocolo, diminuindo-se o nível de complexidade em pequenas frações. Entre cada camada existem interfaces que permitem a ligação entre elas e acesso aos seus serviços, como apresentado na Figura 2.5.

Cada um dos dispositivos LR-WPAN é constituído por uma camada física (PHY), que contém o *transceiver* e os mecanismos que o controlam, e por uma camada de controlo de acesso ao meio (MAC), que controla o acesso ao canal de comunicação. As camadas superiores consistem nas camadas de rede e de aplicação e não estão definidas na norma. A camada física fornece dois serviços: de dados e de manutenção. Esta camada permite o envio/receção de pacotes de/pelo canal transmissão rádio.

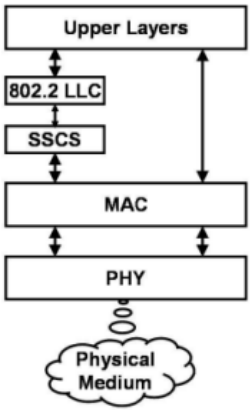


Figura 2.5 – Arquitetura LR-WPAN [2].

A camada MAC providencia dois serviços: de dados e de manutenção. O serviço de dados permite a transmissão e receção de pacotes de nível MAC através do serviço de dados da camada PHY. Esta camada controla o acesso ao canal, *beacons*, o GTS, validação das tramas*, acknowledgments*, associação e saída da rede e fornece também mecanismos de segurança para as aplicações.

2.3.3 Camada física

A camada física possui três versões, as quais se encontram resumidas na Tabela 2.2. A versão europeia só está disponível na Europa e utiliza frequências livres, de acordo com a recomendação *Short Range Devices* (SDR). A versão americana está disponível na América do norte e utiliza uma gama ISM. Utilizam nestes dois casos frequências diferentes devido aos regulamentos dos seus países. A versão Mundial está disponível para uso global e encontra-se numa gama de frequências livres definidas pelo ISM.

Existem no total 27 canais disponíveis nas três bandas: 1 canal para a banda dos 868 MHz, 10 canais na banda dos 915 MHz, e 16 canais na banda de 2.4 GHz. Na versão mais recente do IEEE 802.15.4 de 2006 [2], a camada física foi melhorada. Para as frequências 868/915 MHz, é possível um débito idêntico ao da versão 2.4 GHz (250 kbps).

Tabela 2.2 – Parâmetros da camada física do 802.15.4

Região / Disponibilidade	Europa	América	Mundial
Frequência (MHz)	868 a 868.6	902 a 928	2400 a 2483.5
Nº de canais	1	10	16
Largura de banda	600 kHz	2 MHz	5 MHz
Taxa de símbolos (ksímbolos/s)	20	40	62.5
Taxa de dados (kbps)	20	40	250
Modulação	BPSK	BPSK	O-QPSK

2.3.4 Camada MAC

A camada MAC gere o acesso ao canal rádio e é responsável pelas seguintes tarefas:

- Gerar os *beacons*, caso o dispositivo seja um coordenador PAN;

- Sincronização com os *beacons* da rede*;*
- Suportar a associação e a saída de dispositivos na PAN;
- Suportar segurança no dispositivo;
- Implementar o mecanismo CSMA-CA para acesso ao canal de comunicação;
- Gestão e manutenção do mecanismo GTS;
- Garantir a ligação entre as camadas MAC de dois dispositivos diferentes.

A norma especifica um protocolo MAC híbrido, que pode operar de dois esquemas diferentes para o acesso ao meio:

- Baseado em contenção (*unslotted*): Utiliza CSMA/CA para evitar colisões;
- Baseado em *slots*: Utiliza alocação de *slots* de tempo de transmissão através do mecanismo *Guaranteed Time Slot* (GTS).

Todas as redes baseadas no IEEE 802.15.4 utilizam *beacons* do coordenador quando se juntam à rede novos dispositivos. Num modo normal, uma rede IEEE 802.15.4 pode operar com ou sem o uso regular de *beacons* de comunicação. São apresentadas de seguidas as características de cada modo.

- ***Beacon Enabled***: o coordenador envia periodicamente uma mensagem (*beacon*) com informação acerca do modo de funcionamento da rede, indicação dos *slots* de tempo alocados, o identificador da rede, entre outros. Os dispositivos sincronizam-se com a receção do *beacon* para saberem quando podem transmitir.
- ***Non Beacon Enabled***: não existe sincronização entre os nós e o coordenador, sendo que os nós, quando necessitam de transmitir dados, fazem-no exclusivamente com recurso ao CSMA/CA. Neste modo o coordenador tem de estar sempre ligado para poder receber os pedidos dos dispositivos.

No modo *Beacon Enabled*, o tempo é dividido em duas partes principais denominadas de parte ativa e parte inativa. Na parte ativa, denominada *superframe*, o tempo é dividido em 16 períodos iguais, aos quais dá-se o nome de *slots*. A parte

inativa é opcional e permite que os dispositivos entrem em modo sleep para pouparem energia.

No modo *Non Beacon Enabled*, a estação quando deseja transmitir espera um tempo de *backoff* aleatório e a seguir faz a verificação do CCA (*Clear Channel Assessment*). Se esta for bem-sucedida, envia de imediato os seus dados. Caso contrário, reduz o número de tentativas restantes, espera um novo tempo de *backoff* aleatório e volta a tentar enviar.

2.4 ZigBee

O ZigBee é uma especificação de um protocolo desenvolvido pela ZigBee Alliance, que é uma associação sem fins lucrativos de empresas, grupos de regulamentação governamentais e universidades [3]. O ZigBee foi desenhado como um protocolo genérico para permitir um conjunto diversificado de aplicações. Adicionalmente, foi desenhado para ambientes multi-aplicação e interoperabilidade entre dispositivos de vários fabricantes. As áreas que mais se destacam neste protocolo são o *home automation* e o *smart energy*. Produtos do *home automatition* incluem sistemas de segurança de casas [4], fechaduras eletrónicas, e sistemas de controlo de luzes. Produtos *smart energy* incluem por exemplo sistemas de medição de consumos de energia que permitem aos utilizadores saber quanto estão a gastar.

A primeira versão do ZigBee (ZigBee 2004) foi lançada em dezembro de 2004. Em dezembro de 2006 foi lançada a segunda versão (ZigBee 2006). Em 2007 foi lançada uma nova especificação (ZigBee 2007) que incluí dois modos da *stack*: ZigBee e ZigBee PRO. As especificações que serão aqui apresentadas têm como base a s*tack* ZigBee 2007. A ZigBee Alliance garante compatibilidade entre versões do ZigBee 2007 com versões ZigBee 2006, mas não garante com a versão de 2004.

2.4.1 Protocolo ZigBee

A Figura 2.6 mostra o modelo da *stack* ZigBee. A camada *Security Services Provider* e *ZigBee Device Object* (ZDO) oferecem serviços à camada de rede (NWK -

Network) e de aplicação (APL – Application Layer). Os utilizadores podem desenvolver as aplicações utilizando a *Application Framework* e partilhar o acesso com a *Application Support Sublayer* (APS) e com o serviço ZDO. A camada física (PHY) e MAC do ZigBee são definidas pela norma IEEE 802.15.4.

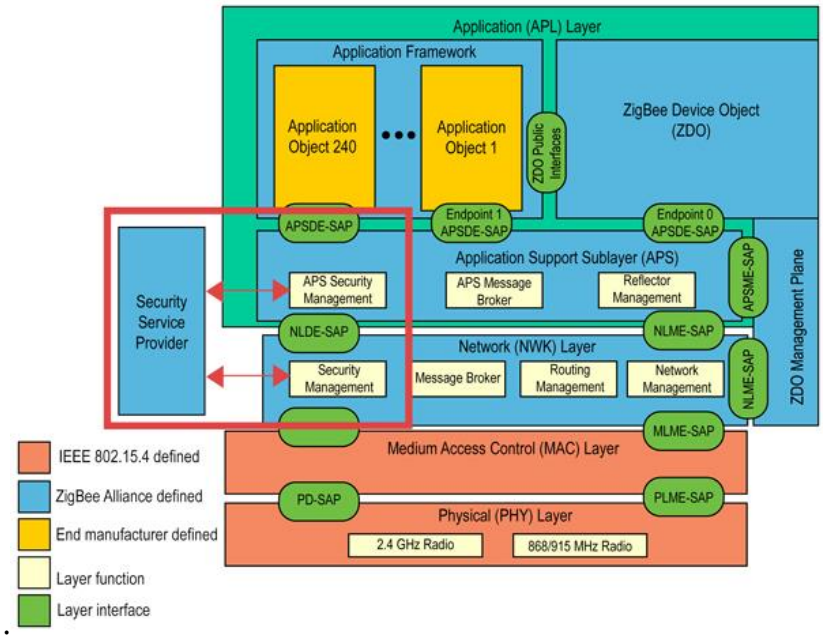


Figura 2.6 – Arquitetura do protocolo ZigBee [5].

O protocolo ZigBee define os seguintes tipos de dispositivos: coordenador (ZC -Coordenador ZigBee), *router* (ZR - ZigBee Router) e *end device* (ZED - ZigBee End Device). Tanto o ZC como o ZR são FFDs, enquanto os ZED são RFDs. O ZC é o equivalente ao coordenador PAN do IEEE 802.15.4. O ZR pode funcionar como um coordenador IEEE 802.15.4 e é capaz do encaminhamento de mensagens e de aceitar associações de novos dispositivos. Os ZED são sempre os nós terminais da rede porque não podem retransmitir informações de outros nós. Geralmente são alimentados a bateria, sendo portanto crucial a conservação da energia para garantir longevidade durante o ser funcionamento. Para isso, são dotados da capacidade de dormir e poder acordar somente quando acontecer um evento relevante.

2.4.2 Camada de rede

A camada de rede (NWK) é a camada inferior da especificação ZigBee. Esta camada fornece um conjunto de serviços através de duas entidades: *NWK Layer Data Entity* (NLDE) e *NWK Layer Management Entity* (NLME). Os serviços da NLDE podem ser acedidos via NLDE-SAP, e os serviços da NLME via NLME-SAP.

A NLDE é a entidade responsável pelos dados de transmissão dos seguintes serviços:

- Gerar o PDU da camada de rede (NPDU) adicionando o *overhead* do protocolo;
- Transmitir o NPDU para o dispositivo, quer este esteja seja o próximo salto ou o ponto final da comunicação.

A entidade NLME permite serviços de gestão para permitir a interação da aplicação com a s*tack*. Os serviços disponíveis são:

- Configuração de novos dispositivos, incluindo iniciar um dispositivo como ZC ou adicionar um dispositivo a uma rede como ZED ou ZC;
- Iniciar uma nova rede (ZC);
- Entrar, sair ou voltar a entrar um dispositivo numa rede;
- Atribuição de endereços para novos dispositivos que entraram;
- Descobrir, registar, e reportar informação pertencente aos vizinhos que estão a um salto;
- Controlar quando um dispositivo está ativo e por quanto tempo;
- Utilizar mecanismos de encaminhamento diferentes como *unicast*, *broadcast*, *multicast* para trocar dados na rede de forma eficiente.

Existem três modos gerais de comunicações disponíveis: *unicast*, *broadcast*, *multicast.* O modo *unicast* é usado para enviar uma mensagem para um único dispositivo. O modo *broadcast* é usado para o envio de mensagens para todos os dispositivos dentro de um raio. O modo *multicast* é usado para enviar mensagens para dispositivos que pertençam a um grupo *multicast* específico.

A camada de rede regista os estados dos vizinhos com quem tem ligação mantendo um contador de falhas de transmissão que é usado para determinar o estado da ligação. Em caso de falha da ligação, a camada de rede inicia o protocolo de manutenção de rotas.

2.4.3 Camada de aplicação

A camada de aplicação (APL) é a camada superior do protocolo ZigBee e consiste na subcamada APS, na *framework* de aplicação e no ZDO. A APS fornece um interface entre a camada de rede com o ZDO e os objetos das aplicações produzidos.

A *framework* de aplicação contém mais de 240 objetos de aplicação definidos pelo utilizador, sendo cada um identificado pelo seu *endpoint*, de 1 a 240, que permite o desenvolvimento e a identificação de diferentes aplicações no mesmo nó. O *endpoint* 0 é usado para endereçar o ZDO, sendo o *endpoint* 255 usado para endereçar todos os *endpoints* ativos. Os *endpoints* 240-254 estão reservados para uso futuro.

O ZDO é responsável pela gestão do dispositivo na rede e fornecer uma interface ao *ZigBee Device Profile* (ZDP). O ZDP é uma aplicação especial que tem a função de fornecer funcionalidades, como a descoberta, configuração, e gestão de dispositivos na rede. O ZDO está diretamente conectado com a camada de rede e controla quando se forma, entra ou sai duma rede, sendo usado como interface para a camada de rede pelas aplicações.

2.4.4 Formato das tramas

O tamanho máximo de um pacote IEEE 802.15.4 é de 133 octetos, sendo estes divididos entre o *payload* e os cabeçalhos de cada camada, conforme apresentado na Figura 2.7 [2]. Uma trama a nível da camada MAC pode ter no máximo 127 bytes. Destes, 9 a 25 correspondem ao cabeçalho, ficando os restantes disponíveis para *payload*. Para obter o maior débito, o maior pacote deve ser enviado. Os cabeçalhos ZigBee correspondem a 16 octetos, ficando assim disponível um tamanho de *payload*

máximo para a aplicação de 85 a 101 octetos, dependendo do tipo de endereçamento usado. Utilizando ZigBee 2006 e anterior, para o envio de um conjunto de dados maiores que o limite máximo definido do IEEE 802.15.4, era necessário dividir estes dados em frações, e enviar cada fração de cada vez. A partir da especificação ZigBee 2007 é possível o envio de dados maiores, em comunicações *unicast*, utilizando para isso fragmentação [6]. Nesta versão, quando a fragmentação está ativa, a camada APS trata das funções associadas, sendo este processo transparente a nível da aplicação.

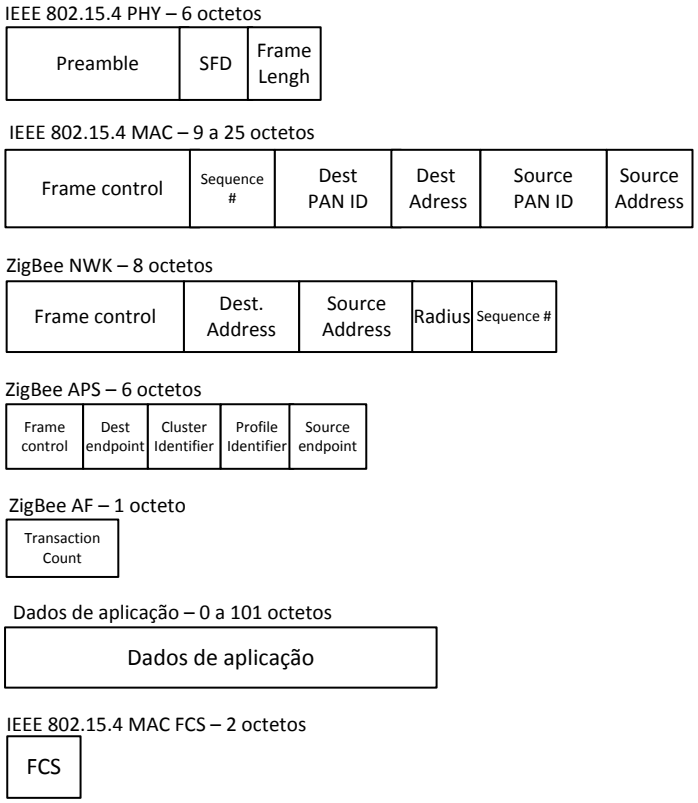


Figura 2.7 – Campos de um pacote de dados ZigBee.

A norma IEEE 802.15.4 incluiu dois modos de endereçamento de dispositivos: um endereço curto, e um endereço longo. O endereço longo consiste num endereço de 64 bits único para cada *transceiver* 802.15.4, e o endereço curto de 16 bits é atribuído a um nó depois de se associar a uma rede. Como a norma IEEE 802.15.4 é designada para LR-WPANs, o uso do endereço curto é aconselhado. A camada MAC reserva 0, 2 ou 8 octetos de tamanho para o endereçamento, conforme seja curto ou

longo [26]. Depois de atribuído o endereço curto, a norma define que ao ser enviado um pacote, se o endereço curto de destino for conhecido, este deve ser usado no pacote preferencialmente em vez do longo. Numa rede ZigBee onde os endereços curtos são conhecidos, são utilizados pelo cabeçalho do MAC de 4 a 16 octetos para endereçamento. Em certos cenários, um dispositivo é membro de múltiplas PANs ou opera em várias PANs na mesma área. A camada MAC consegue isto através do uso de dois campos específicos: *source* PAN ID e *destination* PAN ID. Com isto consegue rejeitar mensagens que não lhe sejam destinadas. Caso os dispositivos estejam na mesma PAN ID, os campos de endereço e destino serão iguais, é só será utilizado o de destino.

2.5 Protocolo SPI

2.5.1 Introdução

O protocolo SPI *(Serial Peripheriral Bus)* consiste numa interface de ligação série síncrona, definida inicialmente pela Motorola [22], que é utilizada para comunicação entre microcontroladores e periféricos. Funciona no modo *full duplex* (dois canais separados, um para envio e outro para receção) e suporta velocidades superiores a 10 Mbps. Utiliza 4 sinais: 2 de controlo (SS – *Slave Select*; CLK - *Clock*) e 2 de dados (MOSI – *Master Output, Slave Input*; MISO – *Master Input, Slave Output*), como apresentado na Figura 2.8

Figura 2.8 – Comunicação SPI.

Por cada impulso de relógio o Mestre recebe um bit de dados no sinal MISO e envia um bit de dados no sinal MOSI. O número de bits a ser transmitidos, palavra, é normalmente 8 bits (1 byte). A transferência de dados é sempre realizada em *full*

duplex, por isso, se se pretender transferir dados num único sentido, o dispositivo no papel de recetor terá que transmitir dados fictícios e o emissor simplesmente ignorar os dados recebidos. Nos microcontroladores que possuem controladores SPI integrados é possível, geralmente, configurar os seguintes parâmetros:

- O número de bits que formam as palavras (e.g., 8, 16);
- A velocidade do relógio;
- Os sinais de seleção dos periféricos - SS;
- O modo de operação – Mestre ou Escravo;
- A fase (CPHA), a polaridade (CPOL) e a ordem.

O protocolo SPI pode ser utilizado com um dispositivo mestre e um ou mais dispositivos escravo. Ao utilizar só um escravo o sinal SS é normalmente mantido a 0 lógico durante o envio de cada palavra enviada. Com a utilização de múltiplos escravos, são necessários sinais SS independentes para cada escravo, como apresentado na Figura 2.9.

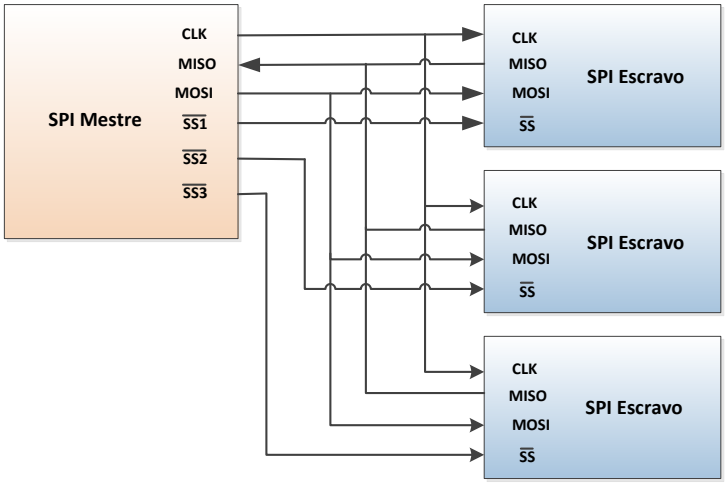


Figura 2.9 – Protocolo SPI com 3 Escravos.

Existem quatro modos de configuração relativos ao posicionamento do sinal de relógio em relação aos sinais de dados. Estes modos definem as combinações correspondentes dos parâmetros CPHA (fase do relógio) e CPOL (polaridade do relógio). Quando CPHA é zero, os dados são capturados na primeira transição de relógio. Quando é um, os dados são capturados na segunda transição do relógio. O

parâmetro CPOL define a polaridade do sinal em repouso e como consequência o sentido das transições durante a transferência. Os dados são sempre capturados pelo recetor na transição de relógio oposta à que são colocados pelo transmissor. Com isto, tem-se sempre meio período de relógio para estabilização de dados, como é possível observar pela Figura 2.10. A ordem dos bits pode ser primeiro o LSB (menos significativo) ou MSB (mais significativo).

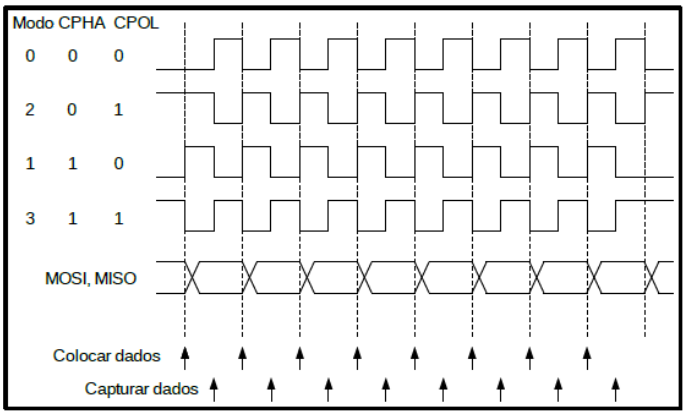


Figura 2.10 – Configuração CPHA e CPOL.

2.5.2 Baud Rate

O *baud rate* do protocolo SPI e UART no módulo CC2530 é dado pela equação (1) [9].

$$Baud\ Rate = \frac{(256 + BAUD_M)\ 2^{BAUD_E}}{2^{28}} f \qquad (1)$$

O $BAUD_M$ e $BAUD_E$ são dois registos que definem o *baud rate* usado para transferências UART e taxa do relógio série para transferências SPI. O valor *f* indica a frequência do relógio do sistema.

A velocidade máxima da comunicação SPI que o CC2530 suporta, utilizando um cristal de 32 MHz, são 16 Mbps ($f/2$) para comunicação unidirecional, enquanto numa comunicação full-duplex a velocidade máxima são 4 Mbps ($f/8$).

A Tabela 2.3 e a Tabela 2.4 apresentam algumas combinações de valores a carregar em cada registo ($BAUD_M$ e $BAUD_E$) para configurar a velocidade da USART,

utilizando um cristal de 32 MHz. Na Tabela 2.4 são apresentados os valores necessários para obter alguns valores do *baud rate* considerados como padrão para as comunicações SPI.

Tabela 2.3 - Valores dos registos para cálculo do *baud rate* SPI.

$BAUD_M$	0	0	0	0	0	0	0
$BAUD_E$	13	14	15	16	17	18	19
Baud rate (bps)	250000	500000	1000000	2000000	4000000	8000000	16000000

A Tabela 2.4 apresenta os valores dos registos necessários para obter alguns valores do *baud rate* considerados como padrão para as comunicações UART.

Tabela 2.4- Valores dos registos para cálculo do *baud rate* UART.

$BAUD_M$	59	59	216	216	216	216	216
$BAUD_E$	7	8	10	11	12	13	14
Baud rate Calculado (bps)	4806,5	9613,0	57617,2	115234,4	230468,8	460937,5	921875,0
Baud rate Padrão (bps)	4800	9600	57600	115200	230400	460800	921600
Erro (%)	0,14	0,14	0,03	0,03	0,03	0,03	0,03

O erro apresentado é referente ao erro relativo associado ao protocolo. O *baud rate* calculado é obtido utilizando a equação (1).

A comunicação SPI pode ser controlada com base em três modos de transferência: verificação do bit de estado (P*olling of Status Bit*), interrupção, e DMA *(Direct Memory Access)*.

3. Descrição e análise do sistema

3.1 Introdução

As redes de sensores sem fios são cada vez mais utilizadas, tanto na indústria como em aplicações domésticas, muito devido ao grande desenvolvimento que têm sofrido nos últimos anos. A sua utilização exige requisitos diversificados de acordo com a necessidade de cada aplicação. Em aplicações sujeitas à transferência de grandes quantidades de dados torna-se necessário um estudo aprofundado sobre os vários protocolos disponíveis e o seu desempenho, com vista a obter o máximo débito possível.

Neste capítulo é feita uma descrição da plataforma utilizada para o desenvolvimento do sistema. De seguida, são apresentados os cálculos teóricos relativos à obtenção do máximo débito possível numa rede em estrela e numa rede em árvore com dois saltos, utilizando as normas IEEE 802.15.4 e ZigBee. Os resultados obtidos nesta análise servem de base para a implementação do RDR.

3.2 Plataforma de desenvolvimento

A plataforma de hardware utilizada é baseada no circuito integrado CC2530, fornecido pela Texas Instruments. Este SoC (*System on Chip*) integra no mesmo chip um microcontrolador e um *transceiver* que permite o desenvolvimento de pequenos dispositivos de sensorização. O CC2530 opera na banda dos 2.4 GHz e oferece uma taxa de débito de 250 kbps, valor máximo definido pelo IEEE 802.15.4. Na Figura 3.1 são apresentados os componentes do kit de desenvolvimento: SmartRF05EB [25], módulo CC2530EM [19], e módulo CC2531 [24].

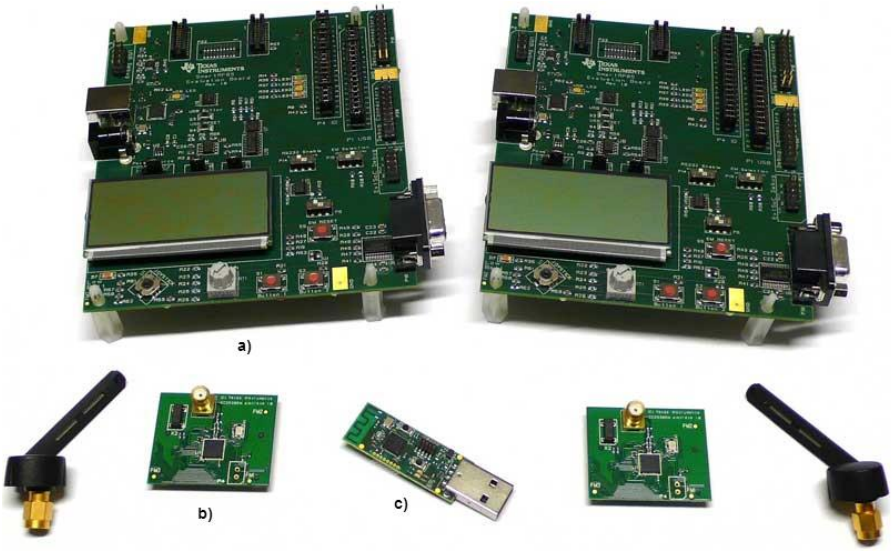


Figura 3.1 – Texas Instruments SmartRF05EB (a), módulo CC2530EM (b), e Modulo CC2531 (c) [23].

O CC2530 inclui um conjunto de funcionalidades que fornecem suporte para a norma IEEE 802.15.4, como a geração automática do preâmbulo da camada física (PHY), a geração e verificação do CRC, um indicador para o CCA através da intensidade do sinal recebido (RSSI), e encriptação / desencriptação AES automática.

O *transceiver* rádio contém um núcleo de processamento para automatizar os procedimentos, em paralelo com o microcontrolador. Este permite receber ou enviar pacotes ao mesmo tempo que o microcontrolador esteja a processar outros pacotes. O *transceiver* rádio faz também o reconhecimento e filtragem dos endereços dos pacotes que recebe de modo a rejeitar pacotes que não sejam dirigidos para aquele dispositivo, diminuindo com isto a carga de processamento do microcontrolador.

O CC2530 inclui um processador de alto desempenho e baixo consumo de energia baseado no microcontrolador 8051, e permite funcionar em 5 modos de operação diferentes de forma a economizar energia. O CC2530 utilizado neste trabalho inclui de 8 kB de RAM e 256 kB de memória *flash*, sendo que existem quatro versões com tamanhos de memória *flash* diferentes: CC2530F32/64/128/256, com 32/64/128/256 kB de memória, respetivamente.

O CC2530 fornece um conjunto de periféricos, destacando-se dos seguintes:

- O temporizador IEEE 802.15.4 MAC (*timer 2*), que é um temporizador dedicado para utilização da camada MAC;
- Um conjunto de temporizadores de uso geral: um temporizador de 16 bits (*timer 1*) e dois temporizadores de 8 bits (*timer 3* e *timer 4*);
- Um temporizador de 32 KHz (*sleep timer*) que é usado para definir o período durante o qual o sistema entra e sai do modo de espera de baixo consumo;
- Um monitor de bateria e sensor de temperatura;
- Um ADC de 12-bit com oito canais e resolução configurável;
- Duas USART com suporte para o modo UART e o modo SPI.

A placa de desenvolvimento SmartRF05EB permite o fácil acesso aos portos do CC2530, permitindo a conexão de vários periféricos como LCD, LEDs, UART, SPI, USB, *joystick,* e botões.

O módulo CC2531 é utilizado como *sniffer*, capturando os pacotes que são trocados entre os diversos dispositivos. Este módulo permite uma fácil depuração, pois permite visualizar graficamente os dados presentes nos pacotes e a sua sequência temporal.

A plataforma de desenvolvimento utilizada para produzir os resultados apresentados nesta dissertação foi a Z-Stack, que é a implementação da *stack* ZigBee fornecida pela Texas Instruments. A versão Z-Stack utilizada neste trabalho foi a Z-Stack-CC2530-2.4.0-1.4.0, que suporta os dois perfis da norma ZigBee 2007: ZigBee e ZigBee Pro. Para a edição de código, depuração e programação dos módulos foi utilizado o software IAR Embedded Workbench for 8051 [27].

3.3 Análise do sistema

Para a implementação do RDR foi feito inicialmente uma análise teórica detalhada do envio de dados numa rede com topologia em estrela e numa rede com topologia em árvore com dois saltos (*multihop)*. Com base na análise teórica

efetuada, foi testado o desempenho de um *router* normal e depois implementado um RDR que permite um aumento de desempenho face à utilização de um *router* normal.

3.3.1 Débito com topologia em estrela

O cálculo do débito numa rede com topologia em estrela é feito através de um modelo teórico e dos tempos associados para a transmissão de um pacote utilizando a algoritmo *unslotted* CSMA-CA da norma IEEE 802.15.4 numa rede em estrela *non-beacon enabled.* Este modelo é apresentado na Figura 3.2. O período de transmissão é constituído por um intervalo aleatório de espera – tempo de *backoff* (T_{BO}), um tempo para alterar do estado de emissão (TX) para receção (RX) – *turnaroud time* (T_{TAT}), um tempo de envio do pacote (T_{pacote}), um tempo para alterar do estado de RX para TX (igual ao T_{TAT}), e um tempo para transmissão da confirmação de entrega – ACK (T_{ack}). O pacote inclui o *payload* e também o *overhead* introduzido pela *stack* ZigBee.

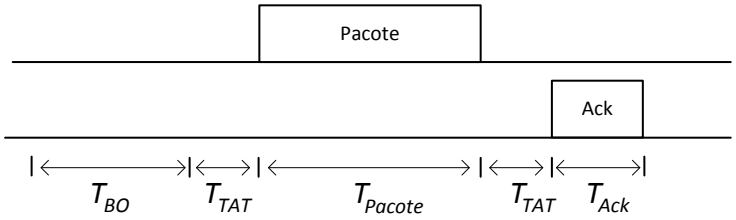


Figura 3.2- Tempos associados ao algoritmo *unslotted* CSMA-CA do IEEE 802.15.4.

O tempo total de transmissão ($T_{total\ Transmissão}$) neste caso é dado pela equação (2).

$$T_{Total\ Transmissão} = T_{BO} + T_{TAT} + T_{Pacote} + T_{TAT} + T_{ACK} \quad (2)$$

O valor do *turnaroud time* (T_{TAT}) é definido pela norma IEEE 802.15.4 e corresponde a 192 µs. O pacote ACK é composto por 11 bytes, sendo o seu tempo de transmissão dado pela divisão do seu tamanho pela velocidade de transmissão 250 kbps.

$$T_{ACK} = \frac{11 \times 8\ bits}{250\ kbps} = 352\ \mu s$$

O tempo de transmissão do pacote (T_{pacote}) pode ser obtido através da equação (3), onde o *Tamanho do pacote* corresponde ao número total de bits transmitidos, que é composto pelo *payload* mais 33 bytes referentes ao *overhead* do protocolo ZigBee.

$$T_{Pacote} = \frac{Tamanho\ do\ pacote}{250\ kbps} \quad (3)$$

O tempo de *backoff* (T_{BO}) é calculado de forma aleatória e depende do expoente de *backoff* e da duração do intervalo de *backoff*. Na primeira iteração do algoritmo CSMA-CA o valor desse expoente está definido como 3, assumindo que o *end device* garante acesso ao meio na primeira iteração (que é verdade se só houver um único dispositivo emissor na rede e não existir fontes de interferências capazes de introduzir erros nos pacotes transmitidos). Isso faz com que o número de intervalos de *backoff* esteja restrito a um valor entre 0 e 7. Um intervalo de *backoff* tem a duração de 20 símbolos, sendo que um símbolo corresponde à transmissão de 4 bits, ou seja, 16 µs, para a velocidade de transmissão de 250 kbps. Portanto, um intervalo de *backoff* equivale a 320 µs. O período médio de transmissão é calculado utilizando o valor médio de *backoff*, que depende do expoente de *backoff*. Multiplicando este valor pelos 3,5 intervalos, que é o valor médio entre 0 e 7, tem-se 1120 µs.

O cálculo do débito máximo útil ($S_{útil}$) – *Goodput* - é feito através da equação (4), em que o *payload* representa o número de bytes dos dados e o $T_{totalTransmissão}$ representa o tempo total de transmissão.

$$S_{útil} = \frac{Payload}{T_{Total\ Transmissão}} \quad (4)$$

Simplificando esta equação para os campos já definidos, obtemos a equação (5).

$$S_{\text{útil}} = \frac{Payload}{T_{BO} + T_{TAT} + T_{Pacote} + T_{TAT} + T_{ACK}} \quad (5)$$

O cálculo do débito máximo bruto (S_{bruto}) – *Throughput* - é feito através da equação (6), onde o tamanho total dos dados incluiu o *payload* e o *overhead* do protocolo.

$$S_{bruto} = \frac{Payload + overhead}{T_{BO} + T_{TAT} + T_{Pacote} + T_{TAT} + T_{ACK}} \quad (6)$$

Com base na equação (5) foi calculado débito máximo útil utilizando um pacote com 90 bytes, sendo este valor próximo do valor máximo para o tamanho de dados de um pacote a nível da camada de aplicação.

$$T_{Pacote} = \frac{(90 + 33) * 8}{250\ kbps} = 3{,}936\ ms$$

$$S_{\text{útil}} = \frac{90 \times 8}{1{,}12 + 0{,}192 + 3{,}936 + 0{,}192 + 0{,}352} = 124{,}31\ kb/s$$

Para a banda 2.4 GHz, a velocidade de transmissão de uma rede IEEE 802.15.4 é de 250 Kbps. A diferença entre este valor e o obtido (cerca de metade) é justificado pelo *overhead* introduzido pelo protocolo (períodos de *backoff,* cabeçalhos de pacotes, etc.).

O valor obtido do débito teórico é referente ao cálculo direto não sendo utilizada qualquer *stack*. Foi calculado o débito utilizando a Z-Stack programada em dois módulos CC2530EM, sendo um programado como *end device* e outro como coordenador, como apresentado no diagrama da Figura 3.3.

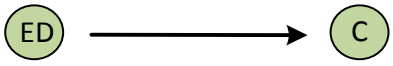


Figura 3.3 – Topologia em estrela utilizada para o teste.

Com este cenário foram efetuados dois testes: modo 1 e modo 2. No teste modo 1 o *end device* gera pacotes, um a seguir a outro, na camada de aplicação e envia para a camada inferior, à velocidade máxima possível, não sendo utilizado qualquer mecanismo de controlo de fluxo. Foram utilizados pacotes com tamanho de 10, 20,

30, 40, 50, 60, 70, 80 e 90 bytes e verificada a variação do débito obtido. Não foram realizados testes com pacotes maiores devido à limitação do tamanho máximo imposto pela norma IEEE 802.15.4. No teste modo 2 é enviado um pacote do *end device* para o coordenador e depois é esperado pelo ACK da camada MAC. Só depois é enviado o próximo pacote. Um diagrama do envio de um pacote e espera do respetivo ACK na camada MAC é apresentado na Figura 3.4.

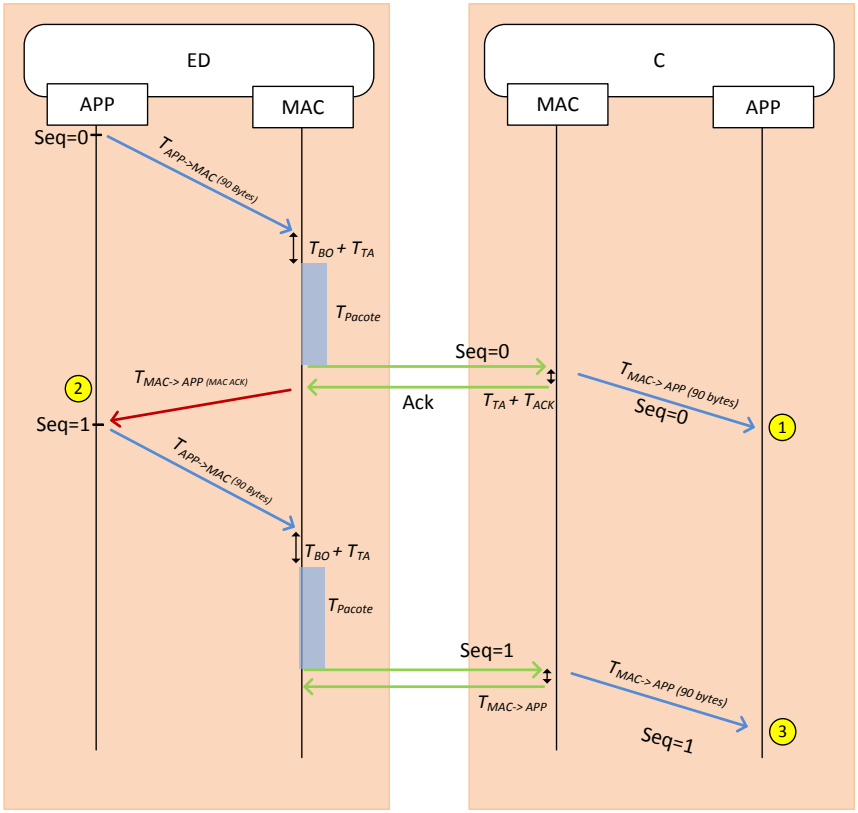


Figura 3.4 - Envio de pacotes por ZigBee no modo 2.

Na figura são apresentados os tempos associados a cada camada durante a transferência de dados. O ponto 1 indica o momento desde que um pacote é enviado da camada de aplicação do *end device* até chegar à camada de aplicação do coordenador. O ponto 2 indica o momento desde do envio de um pacote da camada de aplicação do *end device* até receber o respetivo ACK da camada de MAC. Com o tempo do ponto 3 é possível obter teoricamente o tempo entre envio de pacotes.

Os valores apresentados na Tabela 3.1 são os valores de referência para os parâmetros utilizados nos testes efetuados.

Tabela 3.1 – Parâmetros comuns aos testes efetuados.

Parâmetro	Valor
Numero máximo de *backoff* que o CSMA-CA deve executar até declarar falha no acesso ao canal (*macMaxCSMABackoffs*).	4
Numero máximo de retransmissões permitidas pela camada MAC do 802.15.4 depois de falha na transmissão (*aMaxFrameRetries*).	3
Perfil ZigBee 2007.	ZigBee PRO
Canal IEEE 802.15.4.	11
Modo de endereçamento.	*Unicast*
Número de pacotes recebidos pelo coordenador até acabar o teste	5000
Distância aproximada entre dispositivos (em centímetros)	50

No protocolo ZigBee a mensagem *DataRequest* é utlizada pelo *end device* para perguntar ao coordenador se este tem dados para lhe enviar. Como estas mensagens interferem nos cálculos dos débitos, foram desativadas.

Para determinar o tempo que demora a receber o total de pacotes, para posterior cálculo do débito, foi utilizado um temporizador do CC2530. Inicialmente utilizou-se o timer 1 que é um temporizador de 16 bits, mas como se tinha que processar rotinas de interrupção pois este tem *overflow* a cada 262,14 *ms (4* µs x 65535) e o tempo de transmissão total é bastante superior. Como tal, optou-se por usar o *sleep timer*. Este temporizador utiliza um relógio a baixa frequência de 32,768 kHz, e cada período equivale a 30,5176 µs. Sendo o cálculo do tempo total da ordem de grandeza de segundos, a resolução de aproximadamente 30 µs é suficiente e aceitável, pois não implica imprecisão significativa no cálculo do débito. Para obter o tempo total registaram-se os valores do *sleep timer* quando chega o primeiro pacote e quando chega o último. Faz-se a diferença entre os dois valores e obteve-se o número total de *ticks* durante a receção dos pacotes. O número de *ticks* é multiplicado pelo período do relógio (30,5176 µs), obtendo-se o tempo total. Aplicando este tempo à equação (5) obtém-se o débito.

O resultado dos testes efetuados é apresentado na Figura 3.5. Verifica-se pelos resultados que o valor no modo 1 é menor do que o valor teórico. Para o modo 1, e no caso do envio de pacotes com 90 bytes foi obtido um débito de 94,64 kbps, sendo que o valor teórico obtido foi de 124,31 kbps. Esta diferença deve-se ao atraso entre processamento de camadas, introduzido pelo sistema operativo da Z-Stack quando está a processar tarefas da *Stack*, e também pela latência associada ao funcionamento do microcontrolador. Para o modo 2 com os mesmos 90 bytes foi obtido um débito de 60,48 kbps. Este valor é inferior ao do modo 1 devido ao facto do *end device* ter que aguardar pelo ACK da camada MAC.

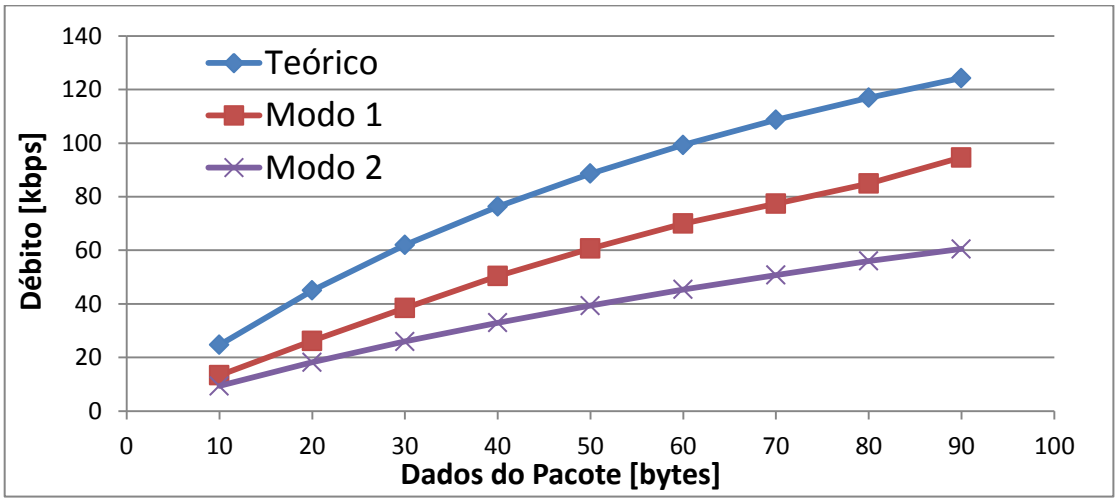


Figura 3.5 - Débito útil máximo teórico e medido numa rede em estrela, no modo 1 e modo 2.

Para questões relacionadas com a qualidade de serviço, torna-se necessário processar o ACK de cada pacote enviado. O ACK pode ser processado na camada de aplicação ou na camada MAC. O ACK da camada de aplicação confirma que o pacote chegou ao endereço de destino, independentemente do número de saltos. O ACK da camada MAC confirma que o pacote foi enviado até ao próximo ponto do destino. No teste efetuado não foi testado o uso do ACK na camada de aplicação pois este exige processamento extra e maior tempo de espera, o que faz com que o débito diminua drasticamente. Por outro lado, no teste modo 1 não há garantia que os pacotes gerados sejam todos enviados e cheguem ao destino.

Com o objetivo de se obter o maior débito possível, foi feita uma análise teórica aos tempos associados ao envio de um pacote de dados de um *end device* para um coordenador com e sem uso da *stack* ZigBee. A Figura 3.6 apresenta um diagrama das camadas para o cálculo do débito sem o uso da *stack* (S_1) e com o uso da *stac*k (S_2). Sem o uso da *stack* os pacotes são trocados diretamente sobre a camada MAC, e com o uso da *stack* os pacotes são enviados a partir da camada de aplicação, para a camada inferior, descendo a *stack*. Do lado do recetor ao pacotes têm que subir a s*tack* até chegar à camada de aplicação.

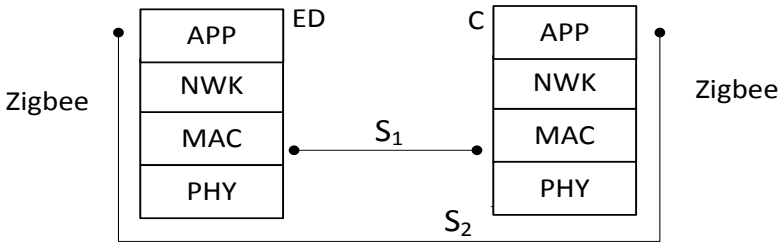


Figura 3.6 – Diagrama de cálculo do débito sem o uso de *stack* (S_1) e com o uso de *stack* (S_2).

O valor do débito S_1 é obtido através da equação (5). Para o cálculo do débito S_2 é necessário adicionar à equação (5) os tempos associados às tarefas de descer ($T_{App->Mac}$) e subir ($T_{Mac->App}$) a respetiva s*tack*. Adicionando os tempos apresentados na figura à equação de cálculo do débito S_2 obtém-se a equação (7).

$$S_2 = \frac{Tamanho\ do\ Pacote\ \times 8}{T_{App \rightarrow Mac} + T_{BO} + T_{TAT} + T_{Pacote} + T_{TAT} + T_{ACK} + T_{Mac \rightarrow App}} \qquad (7)$$

Com base nos valores obtidos em [10], o valor médio medido da descida na s*tack* ($T_{App->Mac}$), para um pacote de 90 bytes, foi de 4,04 ms. O valor medido para a subida ($T_{Mac->App}$) da *stack* para 90 bytes foi de 2,23 ms. Com estes valores é possível calcular o débito teórico do envio de um pacote de 90 bytes a partir da aplicação de um *end device* até este chegar à camada de aplicação do coordenador.

Utilizando a equação (7), para um pacote de 90 bytes obtém-se que o débito teórico ponto a ponto é dado por:

$$S_2 = \frac{90 \times 8}{4{,}04 + 1{,}12 + 0{,}192 + 3{,}936 + 0{,}192 + 0{,}352 + 2{,}23} = 59{,}69\ kb/s$$

O valor teórico obtido é idêntico ao valor obtido nos testes práticos, sendo que podem existir pequenas diferenças dado que o valor teórico é calculado com base no valor médio medido da subida e descida da *stack*. Pelo resultado desta abordagem teórica validam-se os resultados obtidos para o modo 2.

3.3.2 Débito com topologia em árvore

Para uma rede em árvore o débito máximo depende do número de saltos que é necessário para os dados chegarem ao destino. A Figura 3.7 apresenta os tempos associados ao cálculo do débito no caso de uma rede com dois saltos.

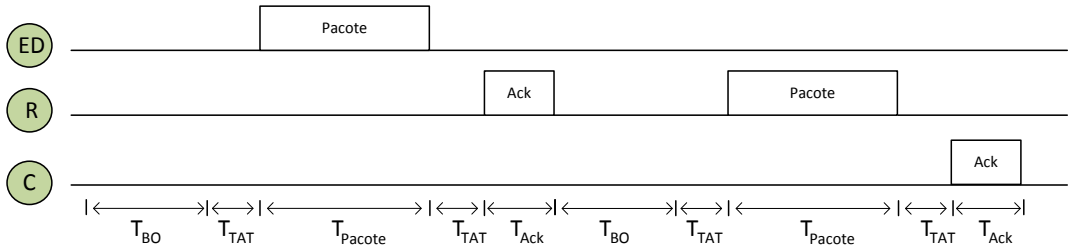


Figura 3.7 - Tempos associados ao cálculo do débito numa rede em árvore com dois saltos.

Para este cenário só estão presentes um *end device*, um *router* e um coordenador. Não havendo mais nenhum dispositivo em simultâneo a transmitir, o débito é calculado com base na equação (7) utilizando para o *Tempo Total de Transmissão* (T_{TT}) o valor obtido da equação (8).

$$T_{TT} = 2 \times (T_{BO} + T_{TAT} + T_{Pacote} + T_{TAT} + T_{ACK}) \quad (8)$$

Para o teste prático em árvore foi utilizada a topologia apresentada na Figura 3.8, onde um *end device* gera pacotes, o *router* recebe e reenvia para um coordenador.

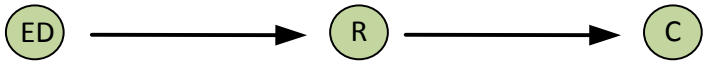


Figura 3.8 - Topologia em árvore utilizada para o teste.

Verificou-se que utilizando a mesma abordagem do teste anterior no modo1, o *router* bloqueava consecutivamente. Com a ajuda do *packet sniffer* foi observado que o *router* reenviava os pacotes apenas por alguns segundos, e depois bloqueava cerca de 8 segundos. Depois deste tempo voltava a reenviar pacotes e o mesmo problema sucedia. Este problema verifica-se devido ao *router* estar a receber grande carga de pacotes e não conseguir enviar os dados para a camada de rede (NWK), visto que a camada MAC está sempre a ser interrompida com a chegada de novos dados, pois tem maior prioridade [10]. Para o teste foi só obtido o resultado com o modo 2, onde o pacote seguinte só é enviado quando receber o ACK da camada MAC do pacote anterior. Com isto o *router* tem tempo suficiente para reenviar o pacote.

O resultado dos testes obtidos com esta topologia são apresentados na Figura 3.9. A discrepância entre o valor obtido e o valor teórico é justificada pelo atraso adicionado pelo processamento entre camadas na Z-Stack.

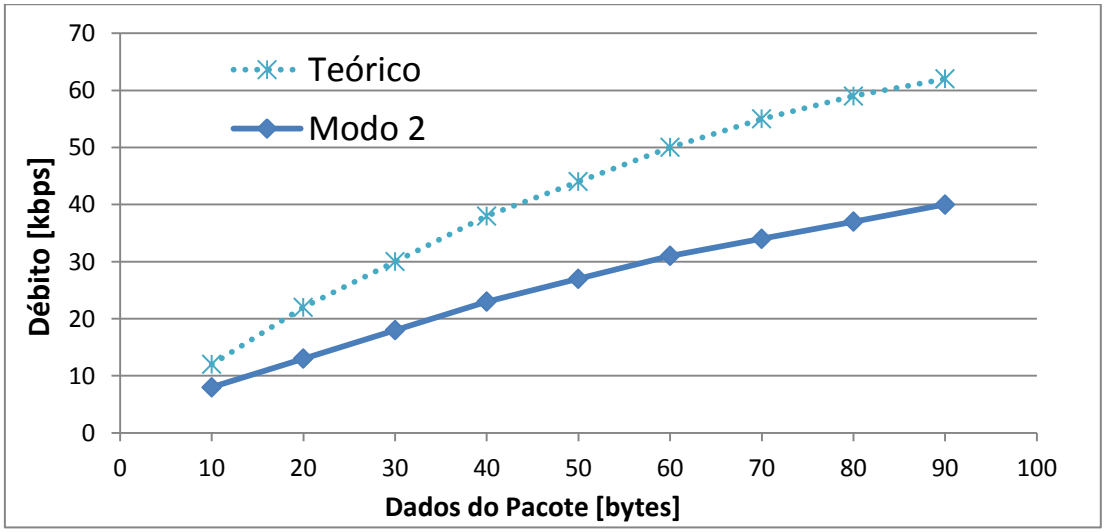


Figura 3.9 – Débito útil máximo teórico e medido numa rede em árvore, no modo 2.

O valor do débito para um pacote de 90 bytes foi de 40,2 kbps. Este valor é inferior ao valor obtido no teste com topologia em estrela (60,48 kbps). Esta diferença é relativa à tarefa do *router* em reencaminhar os pacotes.

Neste teste, e como se usa o perfil ZigBee Pro, este envia periodicamente a mensagem *LinkStatus* para garantir a simetria nas rotas. Estas mensagens foram também desativadas durante os testes efetuados.

3.4 Aperfeiçoamento do débito

Durante os testes anteriores foi analisado o *buffer* que recebe as tramas na camada MAC, que serão depois enviadas para o radio. A monitorização do estado deste *buffer* não estava disponível diretamente, pelo que através da análise e teste verifica-se, ao se enviar dois pacotes seguidos para a camada MAC, um será processado logo de seguida pela camada PHY, enquanto o outro aguardará no *buffer* pelo ACK do pacote enviado. Com a chegada do ACK da camada MAC, o pacote que estava no *buffer* é logo enviado para a camada PHY. Enquanto o radio o processa, a camada de aplicação envia um novo pacote para o *buffer*.

Este funcionamento é apresentado no diagrama da Figura 3.10, onde é possível identificar o envio inicial de dois pacotes, sendo um logo de seguida processado, enquanto o outro fica a aguardar a chegada do ACK da camada MAC.

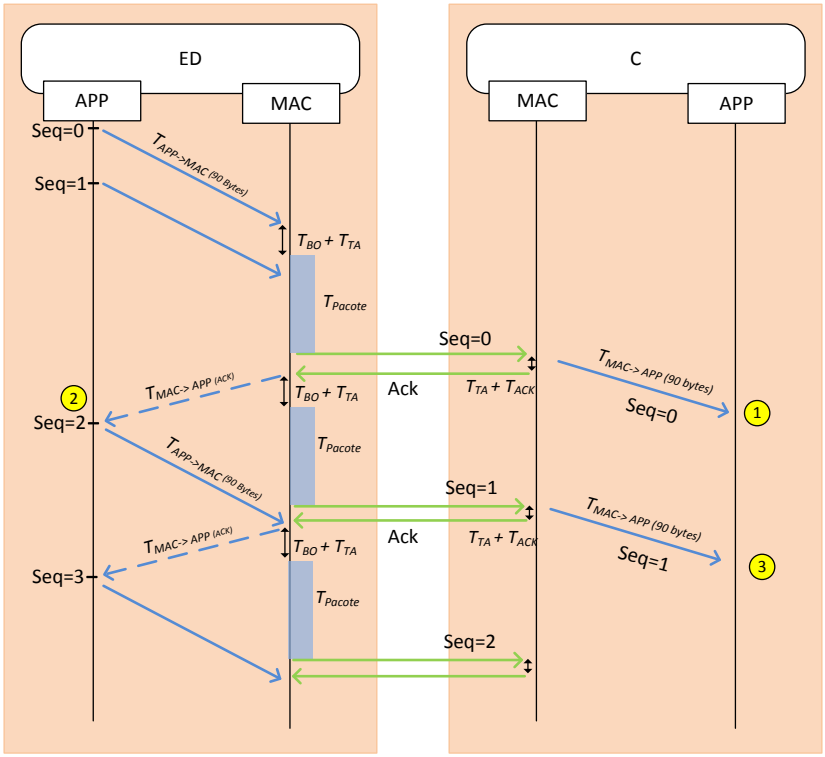


Figura 3.10- Envio de pacotes por ZigBee com envio inicial de 2 pacotes, e espera MAC ACK.

A partir desta análise, foi feita a implementação com a topologia em estrela e realizados os testes do débito. Os resultados obtidos são apresentados na Figura 3.11, onde é feita a comparação com o débito obtido no teste com topologia em estrela no modo 2. Só é apresentada a comparação com o modo 2 pois no modo 1 não é processado o ACK dos pacotes.

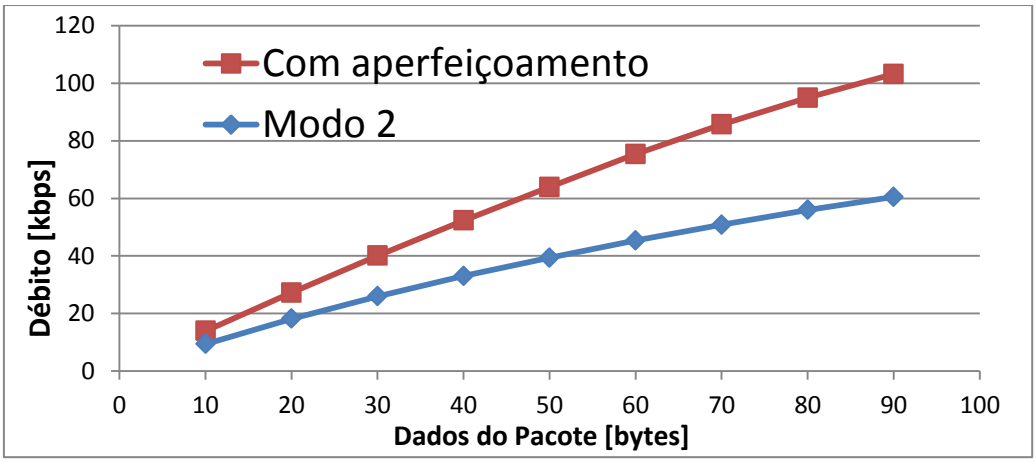


Figura 3.11- Débito útil máximo numa rede em estrela no modo 2 e no modo aperfeiçoado.

Os resultados do teste mostram um melhoramento significativo do débito. Para um pacote com 90 bytes verifica-se um aumento do débito de 70% com o aperfeiçoamento, relativamente aos obtidos no modo 2. Estes valores conseguem ser superiores aos resultados obtidos no modo 1, onde não existe espera por ACK. Pode-se concluir que a constante geração de pacotes na camada de aplicação e posterior envio para a camada inferior, como é feito no modo 1, influência negativamente o desempenho da s*tack*, pois causa interferência entre tarefas, passando estas a demorar mais, originando assim uma diminuição do débito. Verificou-se também que enviando inicialmente mais do que um pacote para o *buffer* não há aumento do débito associado.

4. Comunicação SPI entre módulos CC2530

4.1 Introdução

Neste capítulo é implementado um sistema de comunicação com a interface SPI entre dois módulos CC2530, sendo um configurado como mestre e outro como escravo. Utilizando tramas de 90 bytes, pretende-se testar os 3 modos de processamento SPI (verificação do bit de estado, interrupção, DMA) para determinar o método mais eficiente no que concerne ao débito máximo sobre o canal. Para isso foi adaptado o *baud rate* do protocolo para determinar a velocidade máxima com que se consegue transmitir dados sem falhas. Para testar as falhas foi necessário implementar um mecanismo de deteção de erros, que no caso deste trabalho é baseado em *checksum*. As implementações são programadas diretamente nos módulos CC2530, não sendo usada nenhuma *stack*. Em cada implementação os valores dos débitos são obtidos através da geração e envio continuo de tramas entre o mestre e o escravo, sendo obtido no escravo o tempo total da receção a cada 5000 tramas e enviado pela porta serie, para posterior calculo do débito. A solução mais eficiente é depois utilizada na implementação do RDR.

4.2 Desenvolvimento do sistema

4.2.1 Modo de transferência por verificação do bit de estado

O envio de dados por SPI pode ser controlado com a verificação de bits do microcontrolador que indicam quando determinados dados foram enviados ou quando podem ser lidos. No modo de verificação do bit de estado isto pode ser feito recorrendo aos bits UxCSR.Tx_BYTE e UxCSR.Rx_BYTE ou ao bit UxCSR_ACTIVE.

No caso dos bits UxCSR_Tx_BYTE e UxCSR_Rx_BYTE, no dispositivo mestre, a ativação do bit UxCSR.Tx_BYTE pode ser usada para determinar quando foram enviados dados para o *buffer* UxDBUF. No escravo, a ativação do bit UxCSR_Rx_BYTE indica que os dados do *buffer* UxDBUF podem ser lidos.

Já no caso do bit UxCSR_ACTIVE, no modo mestre, este é ativo quando a transferência de bytes é iniciada, sendo desativado quando termina. No modo escravo, o bit UxCSR_ACTIVE é ativo sempre que o sinal SSN vai a 0, sendo desativado quando este é colocado a 1. Isto faz com que se for utlizada a verificação do bit de estado UxCSR_ACTIVE no escravo, o mestre tenha que colocar a 1 o SSN entre o envio de cada byte transferido.

Para os testes optou-se por só calcular os débitos através da verificação do bit de estado nos bits UxCSR_Tx_BYTE e UxCSR_Rx_BYTE. Os cálculos com o bit UxCSR_ACTIVE têm teoricamente um desempenho idêntico, sendo o débito um pouco menor devido ao facto de se ter que colocar o sinal SSN a 1 entre o envio de cada byte.

Foram efetuadas 3 abordagens diferentes para calcular o débito máximo:

- Teste 1a – O mestre envia tramas com uma estrutura definida para o escravo à maior velocidade possível, sem controlo de fluxo. Inicialmente é utilizado um atraso fixo entre envio de tramas, sendo depois usada uma sequência de alinhamento.
- Teste 1b – Controlo de fluxo por *polling*: São introduzidos comandos para controlar as comunicações SPI. O mestre questiona ao escravo o seu estado. Conforme a tarefa que este esteja a processar, este responde, indicando se pode ou não receber novos dados.
- Teste 1c – Controlo de fluxo por interrupção: É utilizada uma ligação física com o pino P0_7 entre o mestre e o escravo que permite indicar ao mestre quando o escravo acabou de processar o cálculo da deteção de erros. Assim, só quando receber um sinal do escravo, o mestre enviará dados.

4.2.1.1 Teste 1a - Sem controlo de fluxo

Para enviar 90 bytes do mestre para o escravo, foi necessário especificar uma estrutura de trama a utilizar. Como se pretende que não haja perda de dados, cada trama é identificada com um número de sequência. Para detetar possíveis erros na transmissão foi introduzido um sistema de deteção de erros baseado em *checksum*, sendo introduzidos 2 bytes no final de cada trama.

Como método de informar o escravo sobre o número de bytes que irá receber, tornando assim o protocolo dinâmico, o tamanho de cada trama é o primeiro byte a ser enviado. Assim sendo, o escravo sabe por quantos bytes tem que esperar.

Com a adição desses quatro bytes aos 90 bytes iniciais, a trama é composta por uma estrutura de 94 bytes, como apresentado na Figura 4.1.

0	1	2	3	//			93	Bytes
Tamanho	**NºSeq**	**0**	**1**	**2**	**//**	**88**	**89**	**CheckS.Hi**	**CheckS.Lo**	Dados

Figura 4.1 - Estrutura da trama SPI e respetivo tamanho em bytes.

O *NºSeq* é um valor entre 0 a 255 que indica o número de sequência de cada trama. Este valor é incrementado ao fim do envio de cada trama, e serve para o recetor verificar se recebeu os pacotes corretamente.

A trama é preenchida com o valor 0 no byte 1, sendo o valor incrementado até ao byte 89. Isto permite detetar mais facilmente erros durante o processo de depuração (*debug*).

Os últimos dois bytes guardam o valor do controlo de erros (*checksum*) calculado. Este valor é obtido somando byte a byte os valores da trama desde a posição 0 até à posição 91. Depois da soma é feito um *NOT* (complemento para um) e o resultado é colocado pelo mestre nos dois últimos bytes de cada trama a enviar, como mostra na Figura 4.2. Do lado da receção o processo é idêntico, sendo calculada a soma dos valores recebidos de 0 a 90, e feito o *NOT.* De seguida é feita a comparação entre os valores calculados e os dois últimos bytes da trama recebida.

Se forem iguais não houve erros na transmissão. Se forem diferentes, houve erro e os dados são descartados.

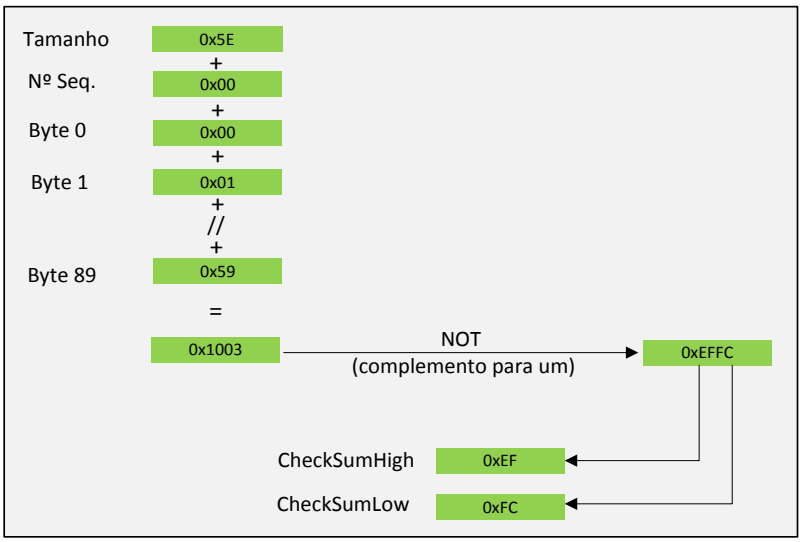


Figura 4.2 - Exemplo de cálculo de *checksum* para uma trama.

Inicialmente foi configurado o SPI a funcionar a 1 Mbps. O mestre envia um byte com o tamanho da trama de dados e depois envia os dados byte a byte. O escravo inicia um temporizador ao receber a primeira trama, e ao fim de 500 tramas recebidas para-o e altera o estado do LED correspondente ao pino P1.0, de modo a se visualizar facilmente o funcionamento da comunicação. Os valores do temporizador são depois enviados pela porta série para o software RealTerm, para permitir calcular o débito. Para a apresentação do tempo seria possível utilizar o LCD mas, como não se está a usar *stack*, seria necessário incluir alguns módulos de software da placa de desenvolvimento o que levaria a que esta tivesse maior processamento. Como se pretende o melhor débito possível, optou-se por usar a porta série.

O protocolo SPI pode ser usado no CC2530 através da USART0 ou da USART1, tendo, para cada uma, duas alternativas de mapeamento de pinos I/O possíveis. Nos testes efetuados foi utilizada a USART0 no mestre, e USART1 no escravo, ficando as ligações como apresentadas no diagrama da Figura 4.3.

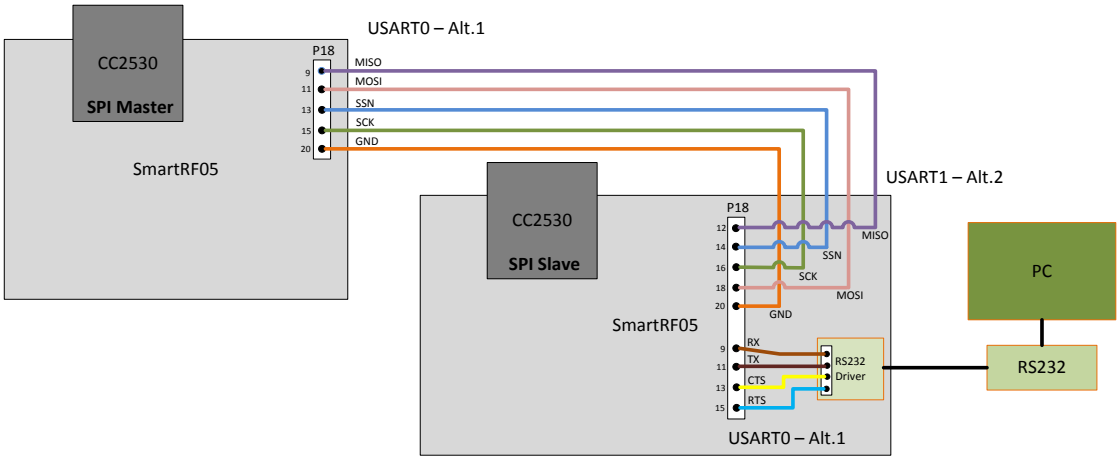


Figura 4.3 - Diagrama de ligação SPI e UART.

O SPI no escravo foi configurado para a USART1 (alt.2) e ficou a USART0 (alt.1) para as comunicações série.

Como os módulos necessitam de tempo para processar os dados, nomeadamente o cálculo do *checksum*, foi necessário inicialmente dar tempo suficiente ao escravo para fazer todos os cálculos antes de o mestre começar a enviar novos dados. Sendo assim, foi introduzido inicialmente um atraso fixo com um valor de 500 µs, após o envio de cada trama, sendo depois calculado o *checksum* da próxima trama.

O diagrama da Figura 4.4 apresenta a estrutura implementada, sendo que T_b é o tempo de envio de 1 byte, *GAP* o tempo entre envio de cada byte, T_d o tempo de atraso gerado no mestre para dar tempo suficiente ao escravo para processar os dados e T_{Ch} o tempo gasto pelo mestre a calcular o *checksum* da próxima trama.

O tempo T_{Ch} indica o tempo necessário para calcular o *checksum,* depois do tempo de atraso fixo definido no mestre. Este valor varia com o tamanho da trama.

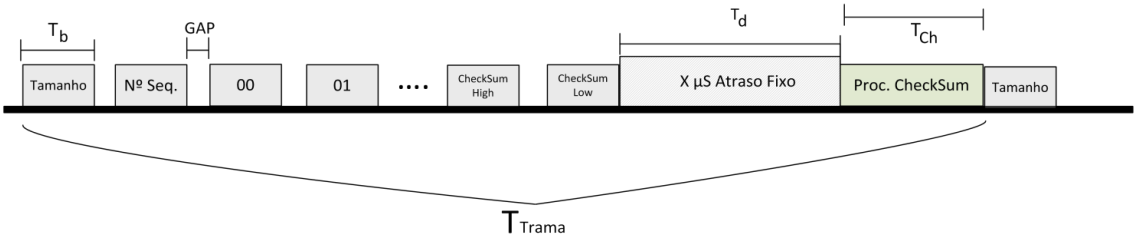


Figura 4.4 - Diagrama de envio de uma trama com atraso fixo e cálculo *checksum*.

O cálculo do débito bruto (*S*) é feito através da equação (9), em que L_p é o tamanho da trama e T_t é o tempo total necessário para transmissão das N_p tramas obtidas nos testes.

$$S = \frac{8 \times L_p \times N_p}{T_t} \tag{9}$$

O débito bruto máximo (S_{max}) teórico é dado através da equação (10), em que L_p é o tamanho da trama e T_{TRAMA} é o tempo total necessário para transmissão duma trama. O débito útil máximo (S_{Util}) teórico é dado pela equação (10), utilizando 90 bytes como tamanho da trama (L_p).

$$S_{Bruto} = \frac{8Lp}{T_{TRAMA}} \tag{10}$$

O cálculo do tempo total do envio de uma trama (T_{TRAMA}) é feito através da equação (11), em que L_p é o tamanho da trama, T_b é o tempo de envio de 1 byte, *GAP* o tempo entre envio de cada byte, T_d é o tempo de atraso gerado para dar tempo ao escravo para processar os dados, e T_{Ch} é o tempo necessário para o cálculo do *checksum*. O tempo de envio de 1 byte é dado pela equação (12), onde R_{SPI} é a velocidade do SPI utilizada.

$$T_{TRAMA} = Lp\, T_b + (Lp - 1)GAP + T_d + T_{Ch} \tag{11}$$

$$T_b = \frac{8}{R_{SPI}} \tag{12}$$

Depois de alguns melhoramentos ao código inicial conseguiu-se reduzir o valor do atraso fixo para 400 µs, sem que haja nenhum erro detetado no envio de dados.

Com a ajuda do osciloscópio verificou-se, como se pode observar na Figura 4.5, que o tempo necessário para a transmissão dos dados de uma trama de 94 bytes a 2 Mbps era de 683 µs (A), e que o valor total de atraso introduzido era de 659 µs (B+C). Para obter o valor de C foi utilizado o Pino P0_7 onde o seu valor é colocado a 1 quando é iniciado o cálculo e colocado a zero quando termina.

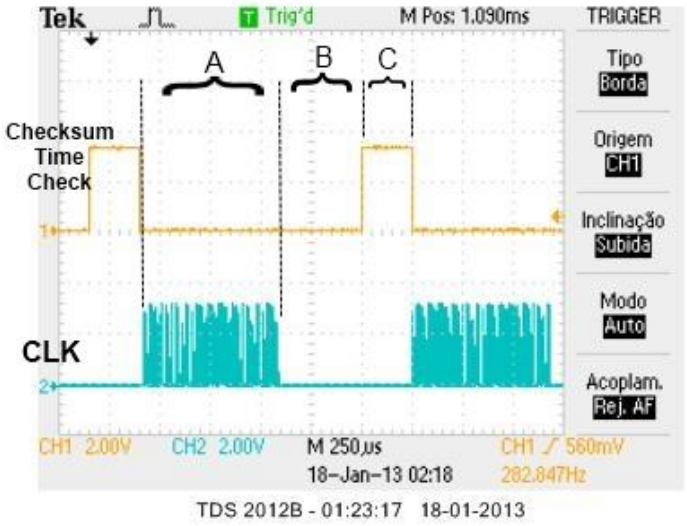


Figura 4.5 – Verificação de tempos obtida com osciloscópio para o caso em que o processamento do *checksum* é feito depois do atraso fixo.

Deste valor verificou-se que 410 µs (B) eram referentes ao atraso fixo programado, sendo 10 µs referentes ao processamento de instruções de configuração do temporizador. Os restantes 249 µs (C) representam o tempo de processamento e de cálculo do *checksum* para os 94 bytes.

Conclui-se, nesta fase, que apesar de o cálculo do *checksum* demorar cerca de 250 µs no mestre, o mesmo cálculo no escravo aparentemente demora mais sendo necessário que haja um atraso fixo adicional mínimo de cerca de 400 µs. Isto pode dever-se ao facto do tempo de processamento de tramas não ser constante (são variáveis e dependem de vários fatores, como a carga de processamento do calculo)

pois o número de tramas recebidas sem erro diminui com o aumento do atraso fixo adicional.

Analisando com o osciloscópio os valores de T_b e *GAP* para as diferentes velocidades SPI obteve-se os valores apresentados na Tabela 4.1.

Tabela 4.1 – Valores de GAP e T_b para diferentes velocidades SPI.

Baud rate SPI	GAP (µs)	Tb (µs)
2 Mbps	3,50	4
4 Mbps	3,33	2
8 Mbps	3,25	1

Perante esta análise, como durante o tempo do atraso fixo o processador do mestre não está a executar nenhuma instrução, o cálculo do *checksum* da trama seguinte foi colocado dentro do atraso fixo, pois o tempo necessário para este cálculo é inferior ao valor do atraso fixo definido. A Figura 4.6 apresenta um diagrama onde está representada esta alteração.

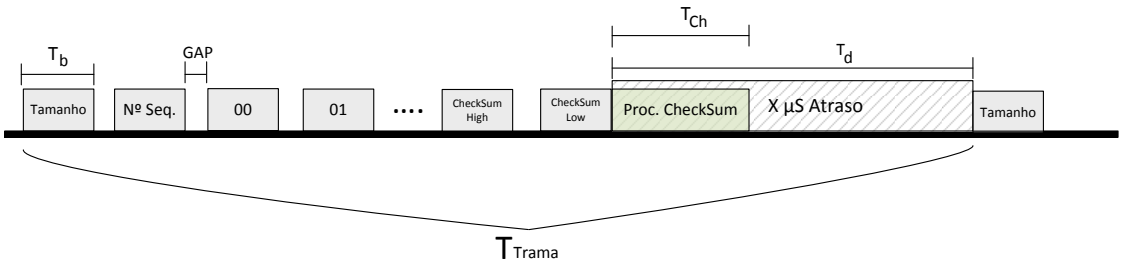


Figura 4.6 – Diagrama de envio de uma trama com atraso fixo.

Face a esta alteração, a equação (11) foi atualizada, passando o tempo total de envio de uma trama a calculado pela equação (13).

$$T_{TRAMA} = N\,T_b + (N-1)GAP + \,T_d \qquad (13)$$

Face à necessidade de um atraso mínimo entre o envio de cada trama de pelo menos 250 µs, relativos ao tempo necessário para processar o *checksum*, foram

realizados novos testes para se obter o valor mínimo do atraso fixo necessário, tendo em vista a obtenção do melhor débito possível, sem erros.

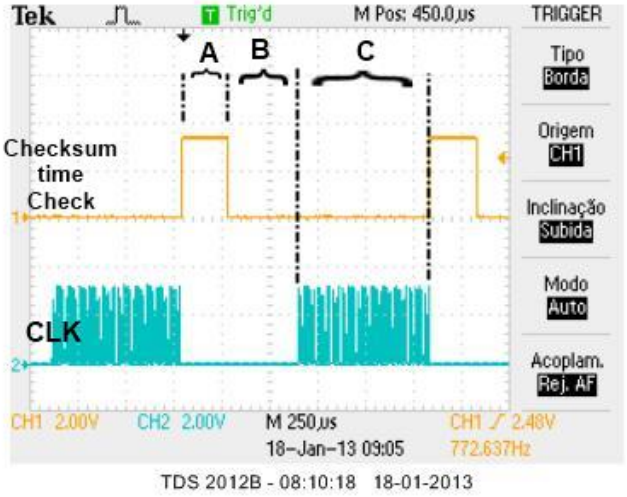


Figura 4.7 - Verificação de tempos obtida com osciloscópio para o caso em que o processamento do *checksum* é feito durante o atraso fixo.

Para esta configuração verificou-se com o osciloscópio, como apresentado na Figura 4.7, que o cálculo do *checksum* da próxima trama demora 246 µs (A), e que o tempo total do atraso introduzido é de 607 µs (A+B), sendo o tempo total de envio de cada trama de 692 µs (C).

Dado que o tempo necessário para o cálculo do *checksum*, obtido através da visualização no osciloscópio, foi de aproximadamente 250 µs, e o atraso mínimo entre tramas sem ocorrência de erros foi de 600 µs, foi necessária uma análise aprofundada para justificar este excesso de tempo em que os módulos não estão em processamento. Verificou-se que o escravo perdia o byte do tamanho em algumas situações. Isto gerava erros pois não era lido o número certo de bytes enviados pelo mestre.

Para resolver este problema foi utilizada uma técnica baseada em sentinela que espera uma sequência definida no início de cada trama. É utilizado um byte (0xAA), definido como *Frame Alignment Byte* (FAB)*, nas* posições 0 e 2 da trama a enviar. A dupla introdução deste byte de alinhamento permite diminuir a possibilidade que surja no meio dos dados uma sequência igual. A estrutura final é apresentada na

Figura 4.8. Com esta estrutura é possível alinhar o escravo para que, em cada trama recebida, este saiba qual é byte inicial e o final, eliminando assim a situação em que o tamanho é lido erradamente a meio de uma trama. Caso uma trama recebida não tenha esta estrutura é ignorada.

0	1	2	3				//				95	Bytes
FAB	**Tamanho**	**FAB**	**NºSeq**	**0**	**1**	**2**	**//**	**88**	**89**	**CheckS.Hi**	**CheckS.Lo**	Dados

Figura 4.8 - Estrutura da trama SPI final.

Com esta alteração verificou-se que o tempo de atraso definido nos testes anteriores não era necessário, pois enquanto o escravo processa o *checksum* da trama recebida, o mestre calcula o checksum da trama seguinte. Como os tempos de cálculo são iguais no mestre e no escravo, é possível enviar tramas seguidas sem tempo de espera adicional. A Figura 4.9 apresenta um diagrama com a estrutura final implementada.

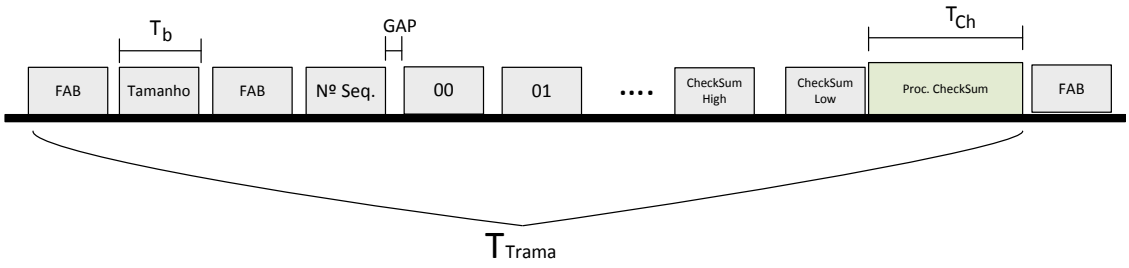


Figura 4.9 - Diagrama de envio de uma trama utilizando alinhamento de trama.

Utilizando o osciloscópio foi possível analisar a implementação e verificar os tempos do processador para as tarefas definidas. Na Figura 4.10 é possível observar o tempo de envio de dados por SPI (A) e o tempo de cálculo do *checksum* (B). O tempo do cálculo do *checksum* é aproximadamente 250 µs.

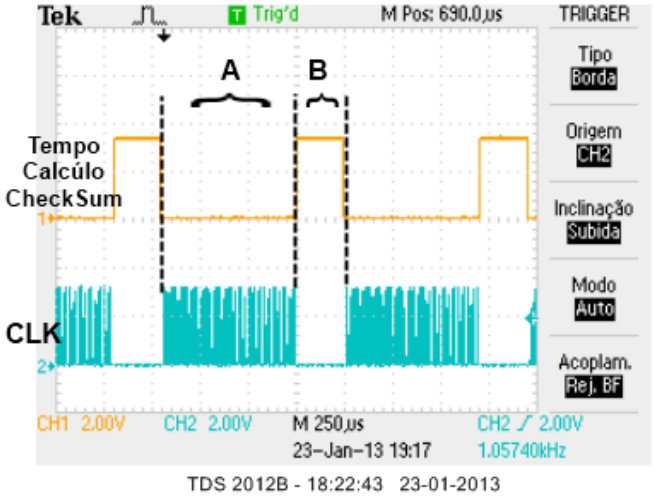


Figura 4.10 - Verificação de tempos obtida com osciloscópio, para configuração com alinhamento de trama e sem atraso fixo adicional.

Face a esta alteração, a equação (11) foi atualizada sendo o tempo de envio de uma trama calculado pela equação (14).

$$T_{TRAMA} = N\,T_b + (N-1)GAP + T_{Ch} \qquad (14)$$

4.2.1.2 Teste 1b - Controlo de fluxo por *polling*.

Como inicialmente no teste 1a teve de se introduzir um atraso fixo para se poder processar os dados, foram estudadas outras alternativas de processar a comunicação via SPI. Uma solução possível e alternativa à solução final do teste 1a é a introdução de um sistema de controlo de fluxo. Com um controlo de fluxo garante-se que a trama seguinte não é enviada enquanto o escravo não estiver pronto, qualquer que seja o atraso necessário. O controlo de fluxo, por ser dinâmico, tem duas vantagens sobre o atraso fixo:

- Permite um atraso menor do que o atraso fixo, aumentando o débito;
- Permite um atraso maior que o atraso fixo, evitando erros.

O mecanismo de controlo de fluxo implementado permite uma gestão dinâmica das mensagens entre módulos, controlando a ocorrência de erros e o pedido de reenvio de tramas.

Na implementação foram definidos comandos para identificar os diferentes estados de cada módulo de modo a permitir que o mestre só transmita uma trama quando o escravo informar que tem disponibilidade para a receber. A Figura 4.11 apresenta um fluxograma com os estados definidos para o mestre. Como a comunicação é *full duplex* o mestre ao enviar um comando recebe logo uma resposta do escravo. O mestre é iniciado com o estado READY que informa o escravo que está pronto para enviar nova trama. Caso o escravo esteja disponível, responde OK, e o mestre envia de seguida uma trama completa de 94 bytes (de acordo com a estrutura da Figura 4.1). Depois do envio é guardada uma cópia da trama e, enquanto o escravo calcula o *checksum* da trama recebida, o mestre calcula o *checksum* da próxima trama. Caso o escravo tenha recebido uma trama com erros, informa o mestre enviando a resposta NOK. Este volta a enviar a trama anterior que ficou guardada. Se o mestre não receber nenhum destes comandos fica a fazer *polling* a cada 1ms até obter resposta da parte do escravo.

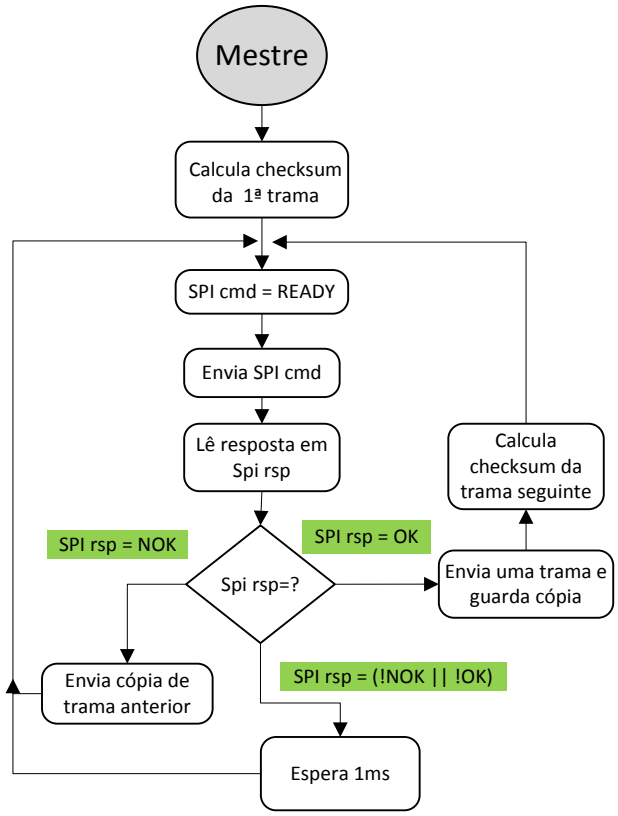


Figura 4.11 - Fluxograma com comandos e estados do mestre.

O escravo inicialmente é definido com o estado pronto a receber dados (OK). Quando receber um comando do mestre vai verificar se é o READY, e se for, recebe uma trama completa. Depois calcula o *checksum* e, caso não hajam erros, verifica se já chegaram as 5000 tramas definidas para calcular o tempo total. Caso hajam erros, é atualizado o byte de resposta NOK, que será enviado na próxima vez que o mestre perguntar o seu estado. Deste modo o mestre só obtém respostas do escravo quando este terminar os processos em curso e estiver pronto para processar novos dados. Um fluxograma com os estados do escravo é apresentado na Figura 4.12.

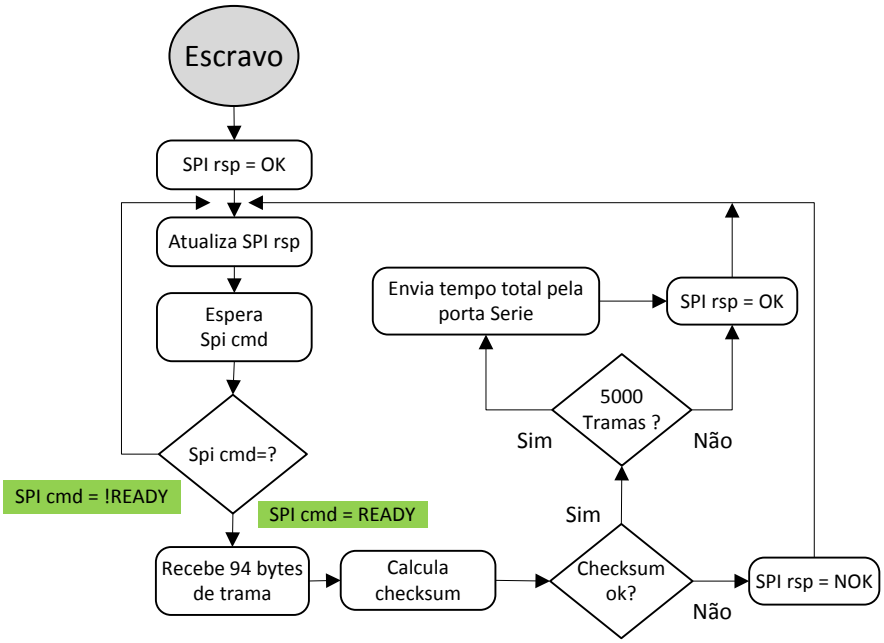


Figura 4.12 - Fluxograma com estados e comandos do escravo.

4.2.1.3 Teste 1c - Controlo de fluxo por interrupção.

Outra forma para implementar o controlo de fluxo entre o mestre e o escravo é utilizando uma interrupção num pino. No mecanismo implementado, o mestre fica a aguardar uma transição no pino P0_7 e quando esta ocorre processa a interrupção enviando os dados por SPI. No final do envio calcula o *checksum* da próxima trama e fica à espera de nova interrupção para a sua transmissão. O escravo gera a interrupção no mestre, colocando o seu pino P0_7 a 1, e recebe os dados por SPI, calculando de seguida o *checksum.* Quando acaba o cálculo volta a gerar nova interrupção para indicar ao mestre que já está disponível para receber outra trama. A Figura 4.13 apresenta o diagrama de ligação SPI entre mestre e escravo, e a ligação da interrupção.

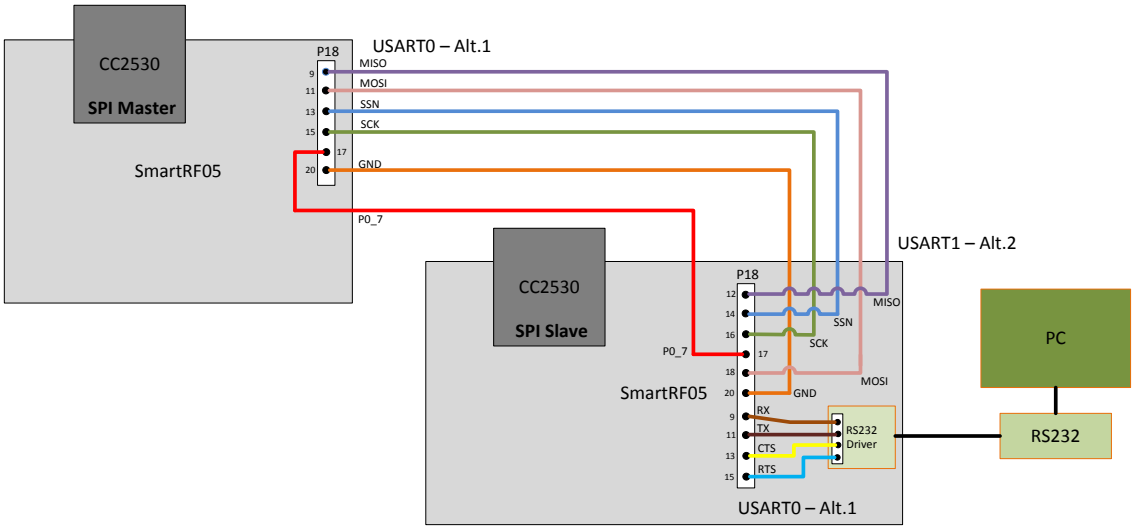


Figura 4.13 – Diagrama de ligação entre mestre e escravo com controlo de fluxo por interrupção no pino P0_7.

A interrupção funciona como um *trigger,* ativando a transferência de dados do SPI. Na Figura 4.14 pode ver-se a interrupção gerada no pino P0_7, a transição do relógio para o envio de dados por SPI (CLK) e o tempo de envio entre cada trama.

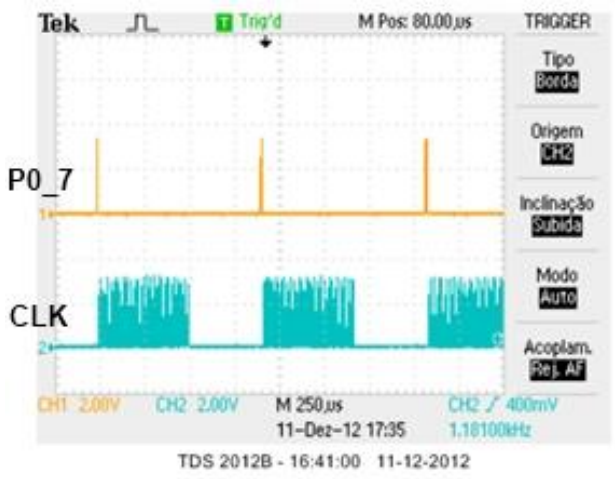


Figura 4.14 – Verificação de tempos obtida com osciloscópio, da configuração com controlo de fluxo por interrupção no pino P0_7.

4.2.2 Modo de transferência por interrupção

Como o CC2530 poderá ter várias tarefas para processar, torna-se necessário garantir que o envio de dados por SPI não seja uma tarefa que ocupe sempre o tempo todo do processador, não deixando tempo para realizar as outras tarefas. As soluções implementadas nos testes 1a, 1b e 1c fazem uso exclusivo do processador, especialmente as tarefas do escravo, pelo que este requisito não é satisfeito. Sendo assim, a transferência de dados por SPI foi adaptada para funcionar por interrupção no lado do escravo, mantendo-se o modo de transferência por verificação do bit de estado do lado do mestre. A implementação de interrupções para a transferência de dados SPI no mestre não é possível devido a problemas conhecidos relacionados com a *flag* de interrupção do CPU USARTx Tx [9].

Uma interrupção é um sinal para o processador gerado por hardware ou software indicando um evento que precisa de atenção imediata. As interrupções funcionam por hierarquias, sendo que, dispositivos com maior prioridade, podem interromper tarefas de dispositivos com menor prioridade. O processador responde suspendendo a tarefa atual, gravando o estado atual, e executando a rotina de interrupção associada. Depois de processada a rotina de interrupção, volta à tarefa que estava anteriormente.

No dispositivo escravo a *flag* da interrupção do CPU USARTx Rx é colocada a 1 quando existem dados disponíveis no *buffer* (UxDBUF). Logo que isto aconteça, o processador salta para a respetiva rotina de interrupção, caso não esteja a executar uma tarefa com maior prioridade. Na rotina de interrupção processa os dados.

Para os testes no modo de transferência por interrupção foram implementadas duas soluções:

- **Teste 2a -** O escravo processa as interrupções byte a byte.
- **Teste 2b -** O escravo salta para rotina de interrupção no primeiro byte de cada trama e recebe os restantes bytes dentro da rotina, por verificação do bit de estado.

4.2.2.1 Teste 2a – Processamento byte a byte por uma interrupção.

Nesta implementação o mestre foi programado para quando receber uma interrupção no pino P0_7 começar a enviar uma trama. Depois de a enviar, calcula o *checksum* da próxima trama e aguarda nova interrupção no pino. O escravo altera o valor do pino P0_7, que faz gerar a interrupção no mestre, para indicar ao mestre que está pronto a receber dados. Quando o mestre envia um byte é gerada uma interrupção no escravo para o processar. Depois de receber uma trama é calculado o *checksum* e no final é alterado o valor do pino P0_7 para indicar que está pronto para receber nova trama. Como o escravo tem que saltar para a rotina de interrupção a cada byte que recebe, foi necessário criar no mestre um atraso com um temporizador de 16 µs, que permita que o escravo consiga receber os dados sem falhas.

4.2.2.2 Teste 2b – Processamento híbrido.

Nesta configuração o escravo salta para rotina de interrupção no primeiro byte de cada trama e recebe os restantes bytes dentro da rotina, por verificação do bit de estado. Esta implementação é idêntica à usada no teste 2a no que concerne ao controlo de fluxo (por interrupção) e ao modo de transferência SPI utilizado para a receção do primeiro byte de cada trama (por interrupção). Ao contrário desta, os restantes bytes da trama são recebidos, dentro da rotina de interrupção, utilizando o modo de transferência por verificação do bit de estado. Sendo assim, o atraso de 16 µs entre o envio de cada byte introduzido no mestre no teste anterior não é necessário.

4.2.3 Modo de transferência por DMA

O CC2530 tem implementado em *hardware* um controlador DMA (*Direct Memory Access*) que permite que sejam movidos dados de uma posição da memória para outra com intervenção mínima do processador. Este pode ser utilizado para

transferências de dados via SPI, diminuindo assim o processamento no microcontrolador do CC2530, aumentando a performance e reduzindo o consumo de energia.

Existem 5 canais DMA disponíveis no CC2530 e 31 eventos configuráveis como *triggers*. Com o DMA é possível automatizar vários processos como o de receber ou enviar dados por SPI ou UART deixando assim o processador livre para realizar outras tarefas. Para utilizar o DMA é necessário configurar uma estrutura própria onde são indicados, por exemplo, o número de bytes a transferir, o endereço de origem, o endereço de destino, a prioridade do canal, e o *trigger* do evento ativo.

Para os testes com DMA foram implementadas duas abordagens, ambas sem o uso de controlo de fluxo:

- **Teste 3a** - O mestre envia dados por verificação do bit de estado e o escravo processa-as através do modo DMA.

- **Teste 3b** - O mestre envia os dados no modo DMA e o escravo processa-as através do modo DMA.

4.2.3.1 Teste 3a – Verificação do bit de estado no mestre.

Neste teste foi implementado o processamento SPI por DMA no escravo. Sempre que chega uma trama completa o controlador DMA gera uma interrupção e na rotina de interrupção do DMA é calculado o *checksum* dessa trama. A receção de dados é feita pelo DMA ficando o processador livre para processar outras tarefas como enviar os valores do tempo total pela porta série. Como exemplo, na Z-Stack, onde existem várias tarefas concorrentes a serem executadas pelo processador, o uso da DMA reduz a utilização do processador.

4.2.3.2 Teste 3b - Modo DMA no mestre e no escravo

Neste teste, tanto o mestre como o escravo foram configurados para transferir os dados via SPI utilizando DMA, deixando livre o processador para outras tarefas.

A Figura 4.15 apresenta uma imagem obtida durante o teste de débito utilizando um analisador lógico e o software USBee. Na imagem é possível verificar que o sinal de relógio (SCK) durante a transferência de dados não apresenta GAP.

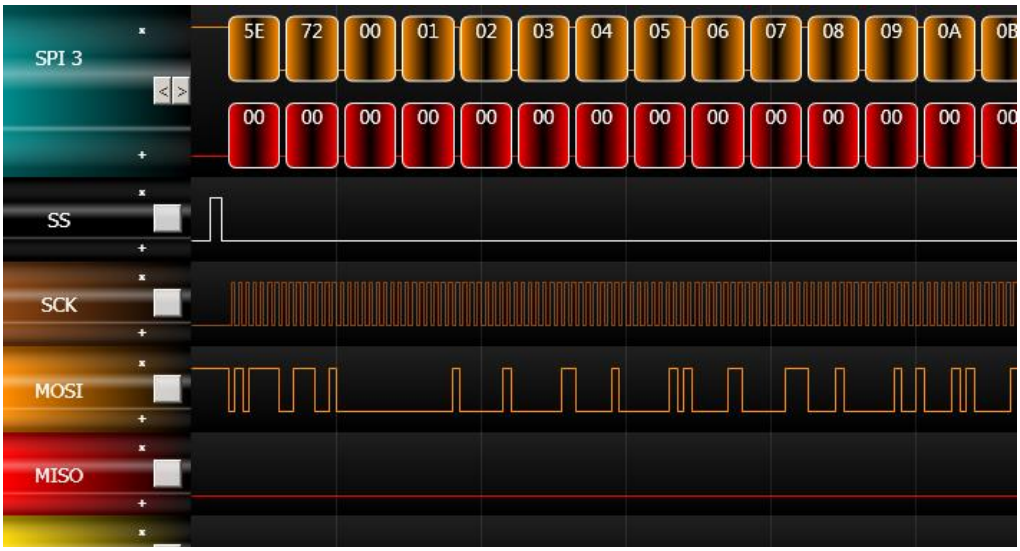


Figura 4.15 - Imagem da captura de dados do SPI obtida utilizando um analisador lógico.

Com a ajuda de um analisador lógico, que tem várias pontas de prova, torna-se bastante mais fácil analisar protocolos com várias ligações físicas, como o caso do SPI, e detetar a causa de possíveis falhas. Neste caso, o indicador SPI 3 corresponde na parte superior aos bytes enviados e na parte inferior aos bytes recebidos. Consegue-se assim identificar cada byte da trama previamente definida.

4.3 Resultados dos Testes

Para receber os resultados do teste pela porta série foram testadas várias velocidades, sendo que 460,8 kbps foi o valor obtido, dentro das velocidades possíveis, que maior velocidade e estabilidade oferecia. Com velocidades superiores verificou-se a presença de vários erros no envio dos dados. Optou-se por usar a

velocidade máxima sem erros pois quanto mais rápido o microcontrolador enviar os dados, mais rápido está preparado para receber novos dados.

4.3.1 Modo de transferência por verificação do bit de estado

4.3.1.1 Teste 1a - Sem controlo de fluxo

Após se ter verificado que a comunicação estava a ser feita corretamente e que não ocorriam erros, foram feitos os testes para diferentes velocidades SPI com e sem cálculo de *checksum*. Os resultados dos testes são apresentados na Tabela 4.2.

Tabela 4.2 - Débitos SPI utilizando alinhamento de trama, sem controlo de fluxo.

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (s)	Débito (kbps)
96	5000	2 Mbps	Não	n/a	3,489	1100,7
96	5000	2 Mbps	Sim	0	4,760	806,7
96	5000	4 Mbps	Não	n/a	2,499	1504,6
96	5000	4 Mbps	Sim	0	3,785	993,3
96	5000	8 Mbps	Não	n/a	2,005	1875,8
96	5000	8 Mbps	Sim	0	3,396	1107,3

Para verificar se os valores dos testes estavam corretos foi analisado no osciloscópio o início de uma trama e calculado o débito teórico com base nesses dados. A Figura 4.16 apresenta um conjunto de bytes do início de uma trama, obtidos no canal MOSI da comunicação SPI entre mestre e escravo. Nos pontos A e C é possível verificar o byte definido como de alinhamento (0xAA). O ponto B representa o tamanho da trama (0x60 = 96 bytes). O ponto D representa o número de sequência que identifica a trama. Os pontos E e F representam o início dos dados com os valores 0 e 1 respetivamente.

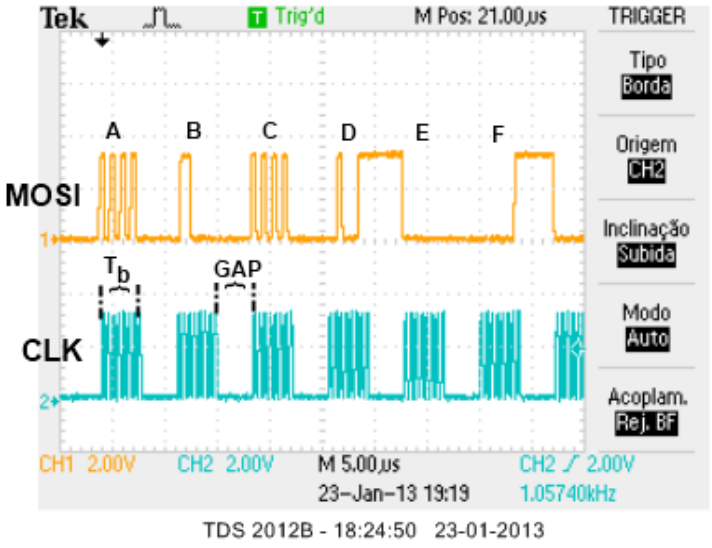


Figura 4.16 – Detalhes da transmissão de uma trama por SPI observada no osciloscópio.

Foi feito um cálculo teórico baseado nos valores lidos do osciloscópio. Com os valores obtidos de T_b = 4 µs (SPI a 2 Mbps) e *GAP* = 3,3 µs, e utilizando a equação (14), sendo T_{Ch} = 249 µs, obtêm-se o tempo de envio de uma trama:

$$T_{TRAMA} = 96 \times 4\ \mu s + (96 - 1) \times 3{,}3\ \mu s + 249\ \mu s = 946{,}5\ \ \mu s$$

Tem-se então um débito teórico baseado na equação (9) de:

$$S = \frac{96 \times 8}{946{,}5\ \ \mu s} = 811{,}4\ kbps$$

Verificou-se que o valor teórico obtido (811,1 kbps) estava bastante próximo do obtido nos testes (806,67 kbps) e confirmou-se assim a validade dos testes.

4.3.1.2 Teste 1b - Controlo de fluxo por *polling*.

Com base nesta implementação foram feitos os testes de débitos para diferentes velocidades SPI e obtiveram-se os valores apresentados na Tabela 4.3.

Tabela 4.3 - Débitos obtidos utilizando controlo de fluxo por *polling*.

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (s)	Débito (kbps)
94	5000	2 Mbps	Não	n/a	3,474	1093,8
94	5000	2 Mbps	Sim	0	4,663	814,9
94	5000	4 Mbps	Não	n/a	2,495	1523,4
94	5000	4 Mbps	Sim	0	3,684	1031,4

Não são apresentados valores para 8 Mbps devido ao CC2530 limitar a velocidade máxima em *full duplex* a 4 Mbps [9]. Como o valor da taxa de erros obtido nos diversos testes é praticamente sempre 0%, o processo de deteção de erros pode ser descartado.

Os dados obtidos com esta implementação não diferem muito dos obtidos com o uso de alinhamento de trama. A principal diferença é que com controlo de fluxo é possível o reenvio de tramas com erros, mas fica limitado só a velocidades de 2 e 4 Mbps. Porém, no método com alinhamento de trama, sem controlo de fluxo, corre-se o risco de ocorrer erros caso o escravo tenha várias tarefas para processar.

4.3.1.3 Teste 1c - Controlo de fluxo por interrupção.

Com esta configuração foram feitos os testes para diferentes velocidades de SPI e obtiveram-se os débitos apresentados na Tabela 4.4.

Tabela 4.4 - Débito SPI com verificação do bit de estado e controlo de fluxo por interrupção.

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (s)	Débito (kbps)
94	5000	2 Mbps	Não	n/a	3,486	1078,7
94	5000	2 Mbps	Sim	0	4,702	799,6
94	5000	4 Mbps	Não	n/a	2,516	1494,2
94	5000	4 Mbps	Sim	0	3,740	1005,4
94	5000	8 Mbps	Não	n/a	2,032	1850,6
94	5000	8 Mbps	Sim	0	3,366	1117,0

Os valores obtidos neste teste são bastante idênticos aos obtidos na implementação com alinhamento de trama. Os valores obtidos para 2 e 4 Mbps são idênticos também aos implementados com controlo de fluxo.

4.3.2 Modo de transferência por interrupção

4.3.2.1 Teste 2a - Processamento byte a byte por uma interrupção.

Os valores dos resultados obtidos com esta implementação são apresentados na Tabela 4.5. Os valores apresentados são bastantes mais baixos dos obtidos nos testes anteriores devido ao facto de o escravo ter que saltar para a rotina de interrupção cada vez que chega um byte. No mestre também foi necessário utilizar um atraso de 16 μs entre o envio de cada byte para o escravo ter tempo suficiente para processar os dados antes que estes sejam alterados.

Tabela 4.5 - Débitos SPI no modo de transferência por interrupção byte a byte .

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (µs)	Débito (kbps)
94	5000	2 Mbps	Não	n/a	1,327	283,4
94	5000	2 Mbps	Sim	0	1,384	271,8
94	5000	4 Mbps	Não	n/a	1,2303	305,6
94	5000	4 Mbps	Sim	0	1,376	273,2
94	5000	8 Mbps	Não	n/a	1,182	318,2
94	5000	8 Mbps	Sim	0	1,328	283,2

4.3.2.2 Teste 2b - Processamento híbrido.

A Tabela 4.6 apresenta os resultados dos testes realizados para esta implementação. Os valores obtidos são bastante próximos aos valores obtidos nos testes 1a, 1b, e 1c. São um pouco inferiores devido a ter que ser processado um salto para a rotina de interrupção a cada trama recebida no escravo.

Tabela 4.6 – Débitos SPI utilizando o processamento híbrido.

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (s)	Débito (kbps)
94	5000	2 Mbps	Não	n/a	3,562	1055,6
94	5000	2 Mbps	Sim	0	4,833	778,0
94	5000	4 Mbps	Não	n/a	2,593	1450,1
94	5000	4 Mbps	Sim	0	3,86	973,1
94	5000	8 Mbps	Não	n/a	2,110	1782,5
94	5000	8 Mbps	Sim	0	3,380	1112,6

4.3.3 Modo de transferência por DMA

4.3.3.1 Teste 3a - Verificação do bit de estado no mestre.

Os resultados obtidos com este teste são apresentados na Tabela 4.7. Os valores de débito obtidos são valores semelhantes aos obtidos nos testes anteriores, com exceção no teste 2a, não havendo variação significativa.

Tabela 4.7- Débitos SPI utilizando o modo de transferência pelo modo de verificação do bit de estado e o escravo processa-as através do modo DMA.

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (s)	Débito (kbps)
94	5000	2 Mbps	não	n/a	3,417	1100,8
94	5000	2 Mbps	sim	0	4,662	806,5
94	5000	4 Mbps	não	n/a	2,447	1536,8
94	5000	4 Mbps	sim	0	3,708	1014,1
94	5000	8 Mbps	não	n/a	1,962	1916,5
94	5000	8 Mbps	sim	0	3,326	1130,5

4.3.3.2 Teste 3b - Modo DMA no mestre e no escravo

Os resultados obtidos nos testes são apresentados na Tabela 4.8. Pelos resultados verifica-se, tal como nos testes anteriores, que o controle de erros

introduz um atraso significativo que faz com que haja uma diminuição do débito. Com este teste conseguiu-se o maior débito dos testes realizados.

Tabela 4.8 – Débitos SPI DMA-DMA.

Bytes	Nº Tramas	Baud rate SPI	Controlo Erros	Taxa Erros	Tempo Total (s)	Débito (kbps)
94	5000	2 Mbps	não	n/a	1,943	1934,6
94	5000	2 Mbps	sim	0	3,188	1179,4
94	5000	4 Mbps	não	n/a	1,533	2452,6
94	5000	4 Mbps	sim	0	2,268	1657,7
94	5000	8 Mbps	não	n/a	1,449	2594,7
94	5000	8 Mbps	sim	0	1,808	2079,2

4.3.4 Tabela final de débitos obtidos

Nos vários testes realizados foram apresentadas as abordagens possíveis para a transferência de dados por SPI. No total foram realizados 7 testes com diferentes modos de funcionamento. A Tabela 4.9 apresenta um resumo dos modos utilizados para obter o débito para cada abordagem.

Tabela 4.9 – Modos utilizados para cálculo do débito em cada teste.

			Modo de Transferência Mestre	Modo de Transferência Escravo 1º byte	Modo de Transferência Escravo Restantes bytes	Controlo de fluxo entre tramas
Polling Stat. Bit:	**1a**	S/ Controlo Fluxo	*Polling*	*Polling*	*Polling*	Não
	1b	C. Fluxo *polling*	*Polling*	*Polling*	*Polling*	*Polling*
	1c	C. Fluxo inter.	*Polling*	*Polling*	*Polling*	Interrupção
Interrupção	**2a**	IRQ - byte a byte	*Polling*	Interrupção	Interrupção	Interrupção
	2b	IRQ + *Polling*	*Polling*	Interrupção	*Polling*	Interrupção
DMA	**3a**	*Polling* - DMA	*Polling*	DMA	DMA	Não
	3b	DMA - DMA	DMA	DMA	DMA	Não

Com base nos resultados obtidos nos testes efetuados, foram reunidos os valores dos débitos por teste e apresentados na Tabela 4.10 para uma melhor comparação.

Tabela 4.10 – Tabela dos débitos geral

			Débitos (Kbps)					
		SPI baud rate	2 Mbps		4 Mbps		8 Mbps	
		Modo	A	B	A	B	A	B
Polling Stat. Bit	**1a**	S/ Controlo Fluxo	806,7	1100,7	993,3	1504,6	1107,3	1875,8
	1b	C. Fluxo *polling*	814,9	1093,8	1031,4	1523,3	n/a	n/a
	1c	C. Fluxo inter.	799,6	1078,7	1005,4	1494,2	1117,0	1850,6
Interrupção	**2a**	IRQ - byte a byte	271,8	283,4	273,3	305,6	283,2	318,2
	2b	IRQ - *Polling*	778,0	1055,6	973,1	1450,1	1112,6	1782,5
DMA	**3a**	*Polling* - DMA	806,5	1100,8	1014,1	1536,8	1130,5	1916,5
	3b	DMA - DMA	1179,4	1934,6	1657,7	2452,6	2079,2	2594,7

A	Com deteção de erros
B	Sem deteção de erros

4.4 Discussão Final

O objetivo deste capítulo consistia em analisar as diferentes abordagens disponíveis para a implementação da comunicação SPI entre dois módulos CC2530 tendo em vista a obtenção do maior débito possível. Face aos resultados obtidos, identifica-se um aumento bastante significativo do desempenho na implementação com DMA no teste 3b. Esta implementação foi assim escolhida para a realização dos testes descritos no próximo capítulo.

Nos testes em que foi usado o *checksum* para a deteção de erros o resultado obtido foi sempre 0%. Como este valor indica que não existiram erros durante a transmissão dos dados, este cálculo poderá ser descartado, aumentado assim o débito e reduzindo o *overhead* do protocolo em 2 bytes.

5. Comunicações ZigBee com o *router dual-radio*

5.1 Introdução

No capítulo 3 foram calculados os débitos máximos para a comunicação ZigBee com um *router* normal, enquanto no capítulo 4 foi obtido o método mais eficiente para a comunicação SPI entre módulos CC2530. Este capítulo debruça-se sobre as comunicações ZigBee utilizando o RDR desenvolvido.

Foi necessário inicialmente implementar, sobre a Z-Stack, um sistema de comunicação com a interface SPI entre dois módulos CC2530 e obter o máximo débito possível. Nesta implementação, o mestre é um coordenador ZigBee. O escravo, por sua vez, é um *end device* (ou um *router*) associado a um segundo coordenador que opera num canal diferente. Com esta implementação, foi feita a comparação com os resultados obtidos nos testes anteriores onde a programação SPI foi feita diretamente nos módulos sem utilização de nenhuma *stack*.

Com os resultados destes testes, foram depois obtidos os débitos de um sistema composto pelos mesmos dispositivos utilizados na implementação anterior. A diferença é que, neste caso, o escravo do RDR envia os dados que recebeu do mestre para o segundo coordenador, utilizando a ligação ZigBee.

No final é apresentado o débito obtido com a implementação do sistema completo, na configuração *End Device* – RDR – Coordenador. Os resultados obtidos com esta configuração são então comparados com os resultados obtidos com o *router* normal, nas mesmas condições.

No âmbito deste capítulo, os testes foram efetuados com o SPI configurado a 2 Mbps pois nos testes realizados no capítulo anterior verificou-se que, com o aumento do baud *rate* do SPI, o aumento do débito não era significativo, sendo este

valor de *baud rate* suficiente para a dar tempo aos módulos para processar os dados. Na Tabela 5.1 são apresentados os valores obtidos nos vários testes do débito máximo entre mestre e escravo, com o SPI configurado a 2 Mbps e programação direta nos módulos. No último teste da tabela foi utilizado o pino P0_7 para controlo de fluxo.

Tabela 5.1- Débitos SPI DMA entre mestre e escravo com programação direta nos módulos.

Topologia	Modo	Baud rate SPI	Deteção erros	Controlo fluxo	Nº bytes	Nº pacotes	Tempo (s)	Débito (kbps)
Mestre-Escravo	SPI DMA	2 Mbps	não	não	94	5000	1,944	1934,6
Mestre-Escravo	SPI DMA	2 Mbps	sim	não	94	5000	3,187	1179,7
Mestre-Escravo	SPI DMA	2 Mbps	sim	sim	94	5000	3,191	1178,5

Verificou-se que o valor obtido com o uso controlo de fluxo é muito próximo do valor obtido sem este. Isto deve-se, neste caso, a que o processador não está ocupado a realizar outras tarefas e demora só mais um pouco do que sem controlo de fluxo, devido ao tempo gasto no processamento da rotina de interrupção utilizada.

5.2 Escolha do escravo do RDR

Antes da implementação do DRR, foram realizados alguns testes para verificar qual dispositivo ZigBee, *end device* ou *router*, seria a melhor escolha para a implementação do escravo do RDR. Para isso, foram testadas duas configurações. Na primeira, um *end device* (ED) gera dados e envia para um Coordenador (C), enquanto na segunda, um *router* (R) gera dados e envia para o Coordenador (C). Os resultados obtidos podem ser observados na Tabela 5.3.

Os testes apresentados podem ser divididos em dois grupos: grupo A,B,C foram realizados com um ED e um C; Os testes D,E,F foram realizados com um R e um C. O

valor obtido no teste A é menor do que o valor teórico devido ao atraso entre processamento de camadas, introduzido pelo sistema operativo da Z-Stack quando está a processar tarefas do sistema.

Tabela 5.2 - Débito máximo teórico obtido ponto a ponto.

Teste / Modo	Topologia	Controlo de Fluxo	Otimização do débito	Tempo (s)	Débito (kbps)
Teórico	-	-	-	3,94	124,31
A	ED → C	não	não	37,98	94,78
B	ED → C	ACK MAC	não	59,38	60,63
C	ED → C	ACK MAC	sim	34,85	103,31
D	R → C	não	não	42,41	84,89
E	R → C	ACK MAC	não	66,75	53,93
F	R → C	ACK MAC	sim	40,74	88,36

Os valores obtidos no teste B e E correspondem ao envio de dados a partir da camada de aplicação para a camada inferior, sendo só enviado o próximo pacote quando for recebido o ACK da camada MAC do pacote anterior.

Os resultados do teste C e F correspondem ao teste aperfeiçoamento do débito, onde são enviados dois pacotes inicialmente para o *buffer* (em vez de um), sendo que o pacote seguinte é processado quando receber o ACK da camada MAC do pacote anterior.

Perante os resultados dos testes pode concluir-se que o ED é capaz de gerar e enviar pacotes mais rápido que o R, pois este está limitado devido às tarefas associadas à gestão da rede que tem que suportar. Sendo assim, o ED foi escolhido para a implementação do escravo no RDR.

5.3 Router Dual-Radio

5.3.1 Implementação e resultados preliminares

Para implementar o RDR foi necessário testar várias topologias de rede para garantir que os dados são encaminhados sem falhas. Foi configurada uma ligação SPI entre um coordenador1 (C1 – mestre) e um *end device* (ED – escravo) para funcionar como *dual router*, e uma ligação ZigBee entre este *end device* e um coordenador2 (C2).

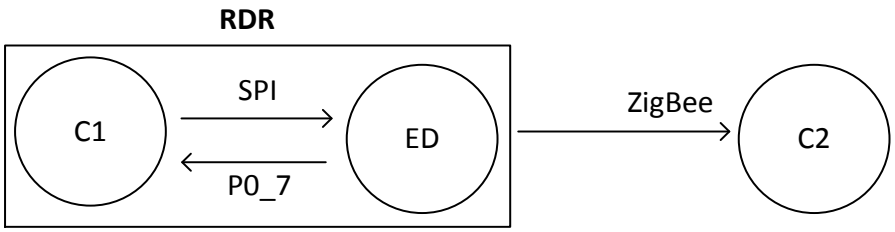


Figura 5.1- Diagrama de configuração para testes SPI com RDR.

Com a configuração apresentada na Figura 5.1, foram feitos vários testes de débito, sendo os resultados apresentados na Tabela 5.3. No teste A o C1 gera constantemente tramas de 94 bytes e envia por SPI para o ED, mas este não reenvia. É calculado o débito baseado no tempo que demora a enviar 5000 tramas. Verificou-se que, do teste anterior efetuado sem *stack* (1178 kbps) para este teste com a Z-Stack (707 kbps), houve um decréscimo acentuado no débito. Isto deve-se ao facto de que as funções de gestão das camadas da Z-Stack necessitam de tempo para serem processadas, diminuindo assim o débito.

Tabela 5.3 - Débitos SPI DMA entre C1 e ED.

Teste	Topologia	Modo	Stack	Baud rate SPI	Deteção erros	Control o fluxo	Tempo (s)	Débito (kbps)
A	C 1↔ED→C2	SPI DMA	Z-Stack	2 Mbps	sim	sim	5,3	707,0
B	C 1↔ED→C2	SPI DMA + ZigBee	Z-Stack	2 Mbps	sim	sim	40,6	92,6

No teste B são reenviados para o C2 por ZigBee os pacotes que o ED recebe por SPI. Neste teste foi verificado que, ao chamar a função AF_Data_Request a cada chegada de uma trama por SPI, esta bloqueava o envio ao fim de pouco tempo devolvendo ZBufferFull(0x11) e / ou ZMemError (0x10). Com uma análise teórica foi verificado que estes erros são causados devido à camada de aplicação estar a tentar enviar dados muito rápido para a camada inferior, não tendo esta possibilidade de resposta.

Para ultrapassar este problema decidiu-se processar o ACK de cada pacote enviado por ZigBee. Para isso só é pedida a próxima trama por SPI (alterado o estado do pino P0_7 para gerar interrupção no C1) depois de o ED receber a confirmação que a trama anterior foi enviada com sucesso para C2. Isto é feito com o processamento do evento AF_DATA_CONFIRM_CMD e respetivo *status* do envio no ED. Os resultados dos testes podem ser observados na Tabela 5.4.

Tabela 5.4 – Débitos SPI DMA entre C e ED, com reenvio por ZigBee para C2.

Teste	Topologia	Baud rate SPI	Deteção erros	Controlo fluxo	ACK	Nº bytes	Tempo (s)	Débito (kbps)
C	C↔ED→C2	2 Mbps	sim	sim	APS	94	115,0	32,6
D	C↔ED→C2	2 Mbps	sim	sim	MAC	94	71,1	52,9
E	C↔ED→C2	2 Mbps	sim	sim	MAC	94	45,2	83,1

O valor apresentado no teste C é obtido utilizando o ACK da camada APS. No teste D o valor de débito é obtido utilizando o ACK da camada MAC. O valor obtido no teste E é obtido com a otimização do algoritmo utilizado em B, sendo inicialmente enviados da camada de aplicação para a camada inferior 2 pacotes, chamando a função AF_Data_Request. Depois deste envio é esperado o ACK do primeiro pacote que foi enviado. Quando este chega é feito outro pedido de dados por SPI (alterado o estado P0_7 para gerar interrupção), sendo estes dados logo reenviados para a

função AF_Data_Request. Com isto garante-se, analogamente à otimização já usada anteriormente, que o *buffer* tem sempre um pacote para processar e não tem de esperar pelo processamento do SPI até chegar o pacote.

Depois de verificado teoricamente o processo referente a cada momento do envio de dados, o RDR foi configurado para receber dados ZigBee. Foi configurada a ligação ao RDR, de modo a permitir que o ED1 gere dados e envie por ZigBee para o C1, estando estes dispositivos configurados num canal. O C1 crie uma trama com os dados recebidos e envie por SPI para o ED2. O ED2 crie um pacote com os dados recebidos por SPI e envia para o C2 por ZigBee, estando ambos configurados num canal diferente. Um diagrama da configuração é utilizada é apresentado na Figura 5.2.

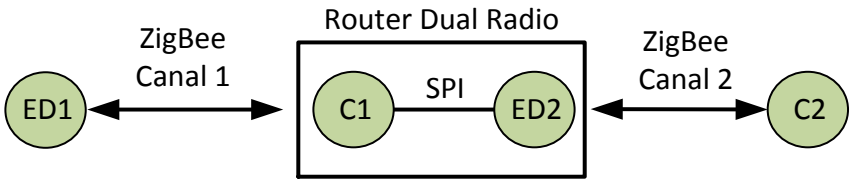


Figura 5.2- Implementação do RDR.

Para a implementação do RDR foi necessário alterar a estrutura de dados ZigBee e SPI. Entre o ED1 e o C1 são utilizadas tramas com 90 bytes que contém um número de sequência no primeiro byte e 89 bytes de informação com uma sequência conhecida, como apresentado na Figura 5.3.

0	1	2	//				89	Bytes
NºSeq	**0**	**1**	**2**	**//**	**85**	**86**	**87**	**88**	Dados

Figura 5.3 – Estrutura de dados do pacote ZigBee entre ED1 e C1.

O C1 quando recebe um pacote ZigBee cria uma estrutura onde coloca o tamanho do pacote, o endereço curto do dispositivo de origem (MSB+LSB), e o *payload* desse pacote, como apresentado na Figura 5.4. Estes dados são obtidos a partir do pacote ZigBee recebido no C1, e são depois acrescentados ao *payload* na trama para envio SPI, ficando esta com um tamanho de 93 bytes. Esta trama é depois enviada para o ED2 por SPI.

0	1	2				//		92	Bytes
Tamanho	**EndHi**	**EndLo**	**Nº Seq**	**0**	**1**	**//**	**87**	**88**	Dados

Figura 5.4 – Estrutura de dados da trama entre C1 e ED2.

Quando o ED2 recebe uma trama por SPI retira o primeiro byte, cria um pacote com os restantes bytes e envia-o para o C2. Como a informação do tamanho do *payload* já vai no cabeçalho da aplicação não necessita ser colocado em duplicado. A estrutura é apresentada na Figura 5.5

0	1	2					91	Bytes
EndHi	**EndLo**	**Nº Seq**	**0**	**1**	**2**	**//**	**87**	**88**	Dados

Figura 5.5 – Estrutura de dados entre ED2 e C2.

Um diagrama temporal com o envio de um pacote utilizando o RDR é apresentado na Figura 5.6. Neste diagrama é possível visualizar as diversas etapas referentes ao envio de dois pacotes, cada um enviado desde um ED para um C por ZigBee. É depois enviado por SPI para o ED2, que o envia por ZigBee para C2. Tanto o envio de dados por ZigBee como de dados por SPI são valores reduzidos, face às restantes etapas do funcionamento do RDR, e foram desprezados na elaboração do diagrama.

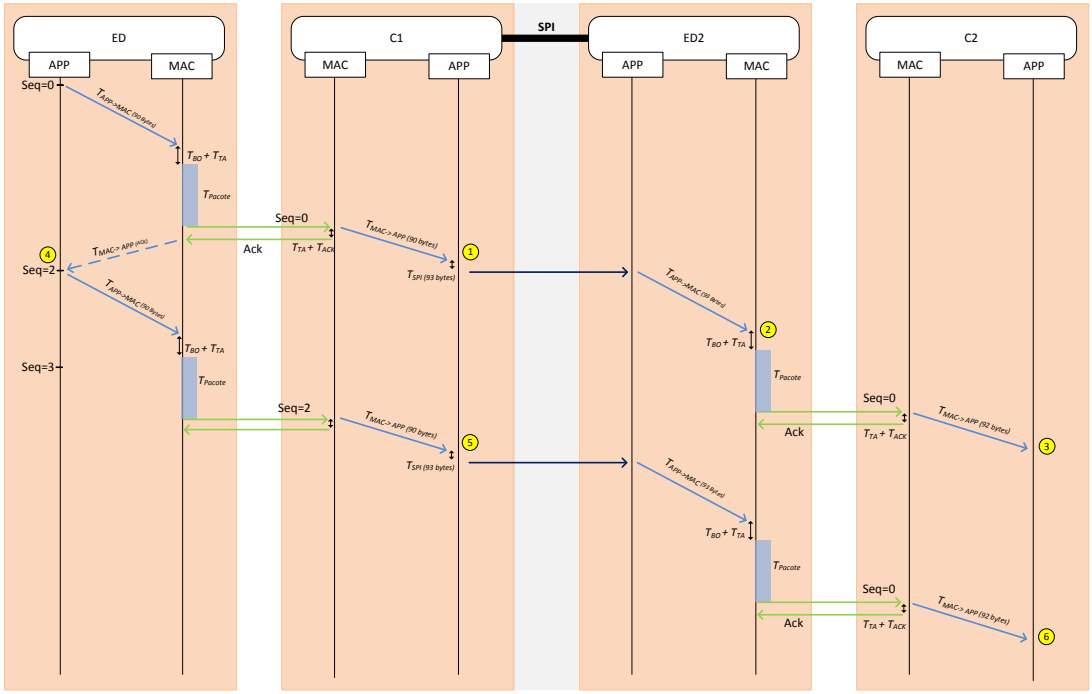


Figura 5.6 – Sequência temporal do funcionamento do RDR.

Na implementação do RDR, foi implementada e testada a validação dos dados durante os testes para garantir que os pacotes chegavam na sequência correta e não eram perdidos, garantindo assim fiabilidade. Para isso, e como cada pacote tem um número de sequência no *payload*, é feito no recetor final um teste de verificação para garantir que não falhou nenhum pacote. No seguimento dos testes elaborados verificou-se que o *checksum* podia ser dispensado, pois os valores deste davam sempre zero. Ao retirar o *checksum*, está a retirar-se overhead que originaria com que o débito fosse menor.

5.3.2 Comparação entre RDR e *router* normal

Depois da implementação do RDR foi feito o teste de desempenho para comparar com o desempenho de um *router* normal. A Tabela 5.5 apresenta o resultado de um teste com um *router* normal e outro com o teste com o RDR.

Tabela 5.5 – Débito obtido com RDR e com *router* normal.

Topologia	Modo	Stack	Nº byte	Nº pacotes	Débito (kbps)
ED → R → C	ZigBee	Z-Stack	90	5000	40,20
ED → RDR → C	SPI DMA + ZigBee	Z-Stack	90	5000	60,75

O valor apresentado para o *router* normal foi obtido anteriormente e apresentados na Figura 3.9. Este valor é referente ao envio de pacotes numa rede em árvore com dois saltos, modo2 (processamento do ACK da camada MAC). No teste com o RDR também foi utilizado o envio com processamento do ACK da camada MAC, criando assim um mecanismo de controlo de fluxo, e garantindo que todos os dados são entregues. Pelos resultados pode-se concluir que a utilização do RDR, no cenário testado, produz um melhoramento do débito superior a 50% em relação ao *router* normal, apesar de introduzir um *overhead* de 2 bytes quando os dados são recebidos por SPI e são depois enviados por ZigBee. O valor do débito obtido com o RDR é semelhante ao valor obtido no teste da rede em estrela, onde um dispositivo transmite dados para o outro, processando o ACK da camada MAC, designado por modo 2 e apresentado na Figura 3.5. Com o RDR o valor foi ligeiramente maior (cerca de 0,5%) do que na ligação em estrela, devido às condições de testes não serem sempre idênticas, nomeadamente interferências no acesso ao meio, temperatura, disposição de dispositivos. Conclui-se, perante os resultados, que RDR funciona como um *gateway* de dados, como esperado, sendo o limite do débito máximo definido pela ligação do primeiro salto.

6. Conclusões e trabalho futuro

Este trabalho tinha como objetivo principal o estudo e avaliação do desempenho de módulos ZigBee, ligados numa rede com topologia em estrela e numa rede com topologia em árvore. Com esse estudo seria implementado um RDR cujo objetivo era apresentar vantagens em relação a um *router* normal, nomeadamente o aumento do débito máximo, além da possibilidade de extensão de protocolos *single-hop* para topologias *multi-hop* e o consequente aumento da área de comunicação da rede.

Inicialmente foram feitos testes de desempenho com o envio de dados entre um *end device* e um coordenador. Seguidamente foram feitos testes com um *end device*, um *router* e um coordenador, sendo os dados gerados no *end device* e enviados para o *router*, sendo estes depois reencaminhados para o coordenador. Durante os testes foi possível registar os débitos máximos para cada topologia utilizada. No caso da topologia em árvore com dois saltos foi verificado inicialmente um comportamento inesperado do *router* onde este bloqueava na retransmissão dos pacotes recebidos, quando sujeito a grande quantidade de dados. Este comportamento foi verificado sem o uso de nenhum controlo de fluxo. Uma possível explicação para este acontecimento poderá ser de como o *router* está sujeito a grande quantidade de dados, a camada de rede não consegue processar todas as respostas pois está constantemente a ser interrompida pala camada MAC, que tem maior prioridade na implementação da Z-Stack. Com a introdução de um mecanismo de controlo de fluxo na camada MAC (ACK) foi possível ultrapassar este problema.

Para a implementação do RDR foi especificado que seriam usados dois módulos CC2530 ligados entre si. Optou-se por conectar os dois módulos através de SPI. O CC2530 dispõe de outras possibilidade de conexão como os portos GPIO e UART, mas o SPI garante maior velocidade de transferência de dados, e quanto mais rápido for a transferência mais rápido fica livre para efetuar outras tarefas. Como o RDR é formado por dois módulos, cada um pode ser programado num canal diferente do

outro. Com isto consegue-se, por exemplo, uma diminuição do número de colisões de acesso ao meio, permitindo assim maior débito. Na implementação do RDR foi utilizado um coordenador ligado por SPI a um *end device*. Inicialmente foi utilizado um coordenador ligado a um *router,* mas verificou-se nos testes efetuados que o *end device* tinha melhor desempenho, pois não tem associado as tarefas de gestão da rede do *router.* Para a implementação do RDR foram feitos testes nos vários modos de processamento do SPI que é possível operar nos módulos CC2530, sendo estes testes realizados com programação direta nos módulos. Com estes testes foi possível comprovar que o modo de transferência de dados DMA permite o melhor desempenho.

Com o modo comunicação entre módulos definido, foram feitos testes com a utilização da *stack* ZigBee da Texas Instruments (Z-Stack). Foram feitos testes onde os pacotes foram gerados no coordenador e enviados por SPI para o *end device,* sendo depois reencaminhados para outro coordenador. Foram detetados problemas de perda de dados e foi necessário implementar um método de controlo de fluxo, utilizando para isso um pino disponível. Com estes desenvolvimentos foi possível colocar em prática o funcionamento do RDR. Para testar o desempenho do RDR foi criado um cenário de testes onde um *end device* gera dados e envia por ZigBee para o seu coordenador, que está configurado num determinado canal. Estes dados são aí encapsulados e enviados por SPI para o *end device* que está configurado noutro canal. O *end device* quando recebe os dados por SPI faz o reenvio para o seu coordenador. Feita a implementação e os testes, obtiveram-se os débitos com o RDR esperados, validando assim o seu funcionamento. Uma limitação que pode seguir no uso do RDR será quando foi atribuído um endereço de rede (16 bits) igual a dois dispositivos que estejam ligado ao mesmo coordenador mas em canais diferentes. Outra limitação deve-se ao facto de os testes serem feitos só num sentido, pois o *end device* em uso no RDR é menos eficiente a receber dados, e seria necessário implementar uma tabela de rotas dos dispositivos ligados ao RDR, para o coordenador saber para onde enviar os dados.

Um possível melhoramento do RDR será adicionar a funcionalidade de o coordenador receber vários pacotes seguidos, fazer a sua agregação numa única trama, envia-lo por SPI para o *end device* que os desagrega e reenvia, um a um, para o outro coordenador. Por exemplo, numa situação onde o coordenador recebe constantemente um pacote de 10 bytes, este pode agregar um conjunto de pacotes e formar uma só trama para depois a enviar por SPI. Este processo é limitado pela capacidade de memória disponível em cada módulo, sendo que também adiciona latência ao envio de cada pacote. Outro ponto a ter em conta é a avaliação do alcance, taxa de entrega, e do consumo de energia com e sem o uso do RDR, sendo também necessário o teste com mais do que um *end device* a transmitir para o coordenador do RDR.

Foi introduzida uma melhoria de desempenho para o envio de dados entre dois dispositivos ZigBee, numa topologia em estrela. São enviados inicialmente dois pacotes da camada de aplicação para a camada inferior, sendo só enviado o próximo pacote quando é recebido o ACK da camada MAC do primeiro pacote enviado. Com isto garante-se que existe sempre um pacote no *buffer* da camada MAC, que permite obter um aumento significativo do débito. Com este desenvolvimento conseguiu-se obter um débito de 103 kbps, sendo este valor bastante superior aos 60 kbps obtidos sem esta alteração. Este valor é também superior aos valores obtidos para o envio de dados sem nenhum controlo de fluxo, que foi de 94 kbps. Uma outra possível abordagem para aumento do débito seria monitorizar o tamanho do *buffer* na camada MAC. Contudo verificou-se que com o envio inicial de vários pacotes para o *buffer* não houve aumento do débito.

No seguimento dos testes efetuados, e pelo volume de dados que as aplicações cada vez mais necessitam de enviar, torna-se interessante a implementação de um RDR onde o reenvio dos dados no segundo salto passe a ser numa rede Wi-Fi. Neste caso o RDR poderia funcionar como um *gateway* onde os dados seriam disponibilizados para qualquer local acessível a partir da Internet. Uma possível solução seria a ligação do coordenador através de SPI a um módulo com ligação Wi-Fi, como por exemplo o GainSpan GS1011M [12]. Este módulo permite um modo

simples, fácil e de custo acessível para adicionar conectividade Wi-Fi a vários produtos, sendo também um módulo de baixo consumo. Com conectividade Wi-Fi, o RDR poderia permitir, com este módulo, débitos até 11 Mbps.

Com o desenvolvimento emergente das tecnologias ligadas às redes de sensores sem fios surgem no mercado cada vez mais soluções de muito baixo consumo, baixo custo e com novas funcionalidades, como é o caso do ZigBee IP [11].O ZigBee IP foi desenhado para equipar a *stack* do protocolo IPv6 em redes de sensores sem fios, funcionando sobre o IEEE 802.15.4, e também atender aos requerimentos do ZigBee Smart Energy 2.0. Esta combinação permitirá aos dispositivos sem fios um maior tempo de vida da bateria. A sua especificação oferece uma arquitetura escalável com ligação IPv6 ponto a ponto baseada nos protolocos de Internet como o 6LowPAN, IPv6, PANA, TCP, TLS e UDP para a criação de uma rede sem fios em malha com eficiência energética e de baixo custo. O ZigBee IP permite a módulos de baixo consumo ligarem-se nativamente com outros dispositivos IPv6 através de Ethernet, Wi-Fi, e HomePlug sem a necessidade de uso de um *gateway* intermediário. O ZigBee Smart Energy 2.0 define a forma como os equipamentos devem comunicar entre si, a ter em conta os aspetos de um ecossistema com energia inteligente. Exemplo deste desenvolvimento é o recente módulo CC2538 [13] da Texas Instruments, que permite o uso tanto da Z-Stack Pro ou ZigBee IP. Este módulo utiliza um microcontrolador ARM Cortex M3 que tem um consumo de energia bastante reduzido, uma grande memória *flash*, interface USB, I^2C, USART e μDMA, entre outros. Estas características permitem que tenha um consumo e desempenho bastante otimizado no contexto das redes de sensores sem fios.

Outro módulo recentemente introduzido no mercado é o GainSpan GS2000 [14], que permite uma grande integração de dispositivos de muito baixo consumo em redes locais sem fios e redes pessoais sem fios. Possui num único chip (SoC) um interface rádio 802.15.4 (ZigBee IP) e 802.11.b/g/n. Com a utilização deste módulo é possível integrar dados, por exemplo, adquiridos numa rede ZigBee, sendo estes depois disponibilizados para uma rede Wi-Fi. O módulo M2MCombo RS9113 [15] da Redpine Signals é também um módulo que permite a integração de vários protocolos

num único chip. Este módulo de baixo consumo possui interface Wi-Fi, ZigBee, e Bluetooth 4.0, e permite que mais do que um protocolo esteja ativo ao mesmo tempo. Esta funcionalidade do módulo permite que seja utilizado como *gateway* entre as diversas interfaces de rádio que possui.

Referências

[1] IEEE Std 802.15.4-2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), 2003.

[2] IEEE Std 802.15.4-2006 – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), September 2006.

[3] ZigBee Alliance. ZigBee Alliance: Wireless Control that Simply Works, 2011.

[4] Alert Me. AlertMe Home Monitoring, 2011. http://www.alertme.com.

[5] Drew Gislason, ”ZigBee Wireless Networking", Newnes, 2008.

[6] ZigBee Alliance, “ZigBee Alliance Document 053474r17” ZigBee Specification v. 1.0 r17, 2007.

[7] WirelessHart, “WirelessHART Overview”, HART Communication Foundation, 2010.

[8] Miwi, “Microchip MiWi™ Wireless Networking Protocol Stack, Microchip, 2010.

[9] Siri Johnsrud and Torgeir Sundet, “CC111xFx, CC243xFx,CC251xFx and CC253xFx SPI DN113 – swra223a”, Texas Instruments, 2009.

[10] Diogo M. F. Gomes, “Modeling and Experimental Performance Analysis of ZigBee/IEEE 802.15.4 for Wireless Body Area Networks”, Tese de Mestrado, Universidade do Minho, 2012.

[11] ZigBee IP, Página web disponível em: http://www.zigbee.org/Specifications/ZigBeeIP/Overview.aspx, consultada em 10-10-2013.

[12] GainSpan GS1011 Datasheet, " GS1011 Ultra Low-Power Wireless System on Chip (SoC)", GainSpan Corporation , 2011.

[13] CC2538 Dataseheet, "A Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN and ZigBee Applications", Texas Instruments , 2013.

[14] GainSpan GS2000 Datasheet, " G2000 Ultra Low-Power 802.11b/g/n + 802.15.4 Single Chip (SoC)", GainSpan Corporation , 2011.

[15] RS9113, "M2MCombo™ 802.11n 1x1, Dual-mode BT 4.0, ZigBee", Redpine Signals, 2013.

[16] J. A. Afonso, L. A. Rocha, H. R. Silva, J. H. Correia, "MAC Protocol for Low-Power Real-Time Wireless Sensing and Actuation", 13th IEEE International Conference on Electronics, Circuits and Systems, Nice, France, December 2006.

[17] I.F. Akyildiz, S. Weilian, Y. Sankarasubramaniam and E. Cayirci, "A survey on sensor networks", in Communications Magazine, IEEE, 40 (8). pp. 102-114, 2002.

[18] Mitsubishi Electric, Página web disponível em: http://www.mitsubishielectric.com/semiconductors/triple_a_plus/technology/02/index2.html, consultada em 20-09-2013.

[19] CC2530 Datasheet, "A True System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications", Texas Instuments, 2011

[20] Wi-Fi Alliance, Página web disponível em: http://www.wi-fi.org/, consultada em 25-10-2013.

[21] Bluetooth Special Interest Group, Página web disponível em: www.bluetooth.org/, consultada em 25-10-2013.

[22] SPI Block Guide V03.06, Motorola Inc, 2000

[23] CC2530 Development Kit, "CC2530 Development Kit User's Guide", Texas instruments, 2010

[24] CC2531 Datasheet, “A USB-Enabled System-On-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee”, Texas instruments, 2010

[25] SmartRF05B Datasheet, “SmartRF05 Evaluation Board User’s Guide”, Texas instruments, 2010

[26] T. Ryan Burchfield , S. Venkatesan , Douglas Weine, “Maximizing Throughput in ZigBee Wireless Networks through Analysis, Simulations and Implementations”,

[27] IAR Embedded Workbench for 8051, Página web disponível em: http://www.iar.com/ew8051/, consultada em 25-10-2013.